Inveo Sp. z o.o.

Proxer

Instruction Manual

Overview

Warning

This manual applies only to the firmware version v0.13.1 onwards. Inveo does not guarantee that the information contained in this document applies to previous firmware revisions.



Purpose of the device

Proxer is a modern line of RFID readers, designed especially for use in trade, logistics, offices, swimming pools or gymnasiums. The visually attractive enclosure and colour LCD display with the option to upload animations and graphics make the Proxer series readers an ideal solution for any modern interior.

The built-in memory for 5000 users and 100 000 logs, as well as a wide array of available communication protocols, make the Proxer a comprehensive solution for access control and work time recording, both as a stand-alone device as well as an element of a larger system.

The readers can be powered both with 12V DC and PoE IEEE 802.3af.

Changelog

1.0 11th of March 2026

- Firmware revision v0.13.1

Table of contents

- Overview
  - Purpose of the device
  - Changelog
    - 1.0 11th of March 2026
- Device construction
  - Technical characteristics
  - Connector description
  - Communication channels and general characteristics
- Network configuration
  - Configuring LAN and Wi-Fi
  - Checking the network status
- Device configuration
  - Device status
  - Administration
    - Access
      - API Keys
    - Time settings
    - Diagnostics, events
    - Adding licenses
  - I/O Settings
    - Relay outputs configuration:
    - Digital input settings:
  - LCD
    - Settings
    - Templates
      - Content
    - Example: displaying text via HTTP Client
  - Predefined variables
  - RFID Settings
    - Setup – RFID standard selection
    - Card data – Only for HF version when Mifare is selected
    - NFC – Only for HF version when Mifare is selected
  - Card Action
    - On-new

        - On-known
        - On-unknown
        - On-hold
        - On-remove
- Managing cards and events
    - Card database
        - CSV upload
        - Downloading the database
    - Event logging
        - Events preview
        - Log download
- Communication protocols
    - Web – HTTP/HTTPS Server
        - HTTP GET – Status
        - HTTP GET – Control
        - HTTP GET – RFID card database
    - HTTP Client
        - HTTP Client configuration
        - HTTP Client – poll request
        - HTTP Client – Card Action event request
        - HTTP Client – Control
    - MQTT Client
        - Publish topic
        - Subscribe topic
    - E-mail
    - Modbus TCP/RTU
        - Holding Registers addresses
        - Single Coil addresses
    - Syslog
    - REST API
    - Lua Scripts
        - Available system functions
        - Lua libraries
            - card library
            - mf (Mifare Classic) library

        - rfid library
        - userdb library
        - dev library
        - table library
        - string library
        - os library
        - math library
        - utf8 library
        - pack library

- Application examples
    - Example 1: Room access control
    - Example 2: Recording of work time
    - Example 3: Automatic lighting control
    - Example 4: Recording of access attempts by unauthorized cards and sending the events to the Syslog server
    - Example 5: Basic access control system using Lua script
    - Example 6: Transfer of device data via HTTP
- Factory settings, backup
    - Restoring factory settings
    - Backup / Restoring user settings
    - Emergency firmware restore
- Warranty and manufacturer's liability
    - Storage, operation and transport conditions
    - Disposal and decommissioning

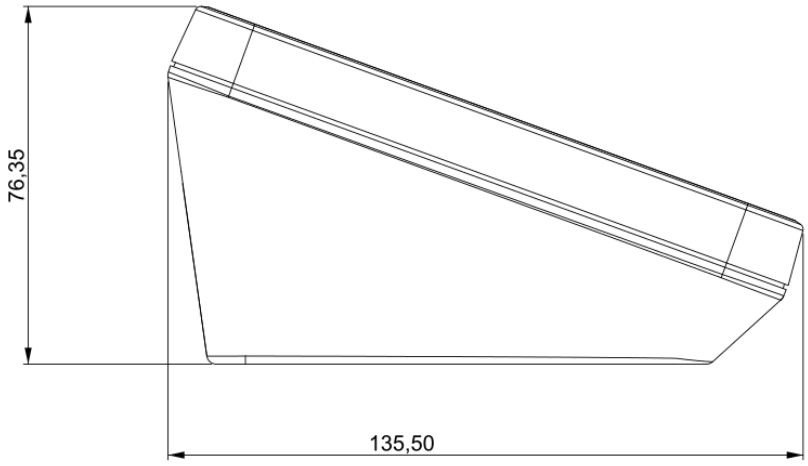
Device construction

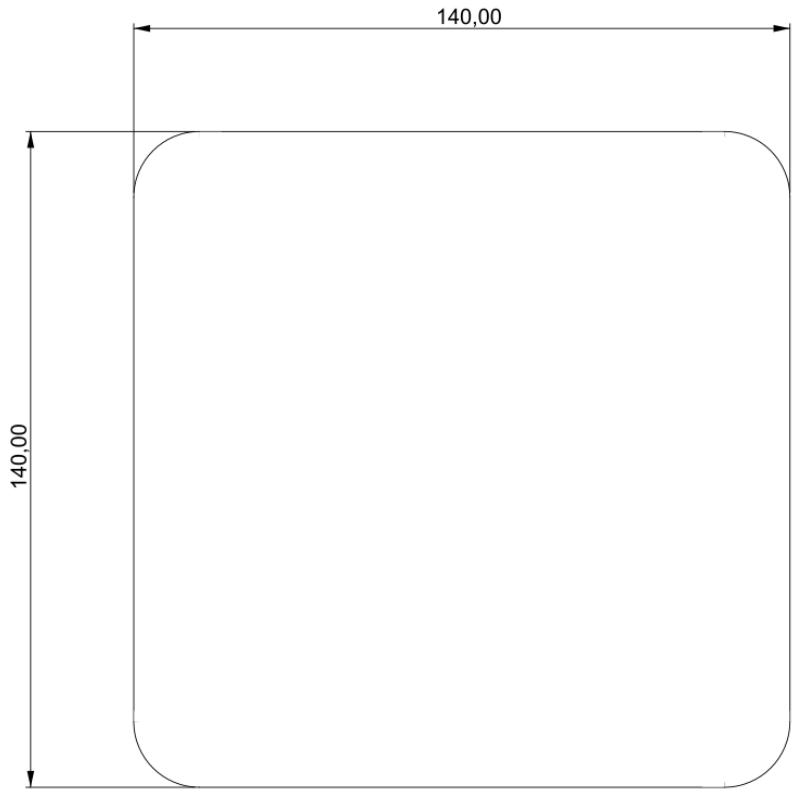
Technical characteristics

Parameter	Description
Power supply	**12V DC** via a quick-connect terminal or 5,5x2,1mm barrel jack **PoE:** 33–57V PoE IEEE 802.3af
Power consumption	**Max. 5W**
Transponders	**The following Proxer versions are available:** - **HF:** High frequency version dedicated for 13,56MHz transponders: Mifare Classic, ICODE, iClass (CSN only). **HF Seos:** Special HF version with full iClass transponder support in addition to functions supported by standard HF. **Built-in memory:** 5000 tags, 100 000 events
Inputs	- **Amount of inputs:** 2 - **Input type:** Normally Open (NO)
Outputs	- **Amount of outputs:** 2 - **Output type:** NO (relay), - **Maximum load:** 1A @ 30V DC
Communication	- **Ethernet** 10/100Mbps Full Duplex, RJ45 port - **Wi-Fi** *(option)*
Enclosure ingress protection	**IP65**

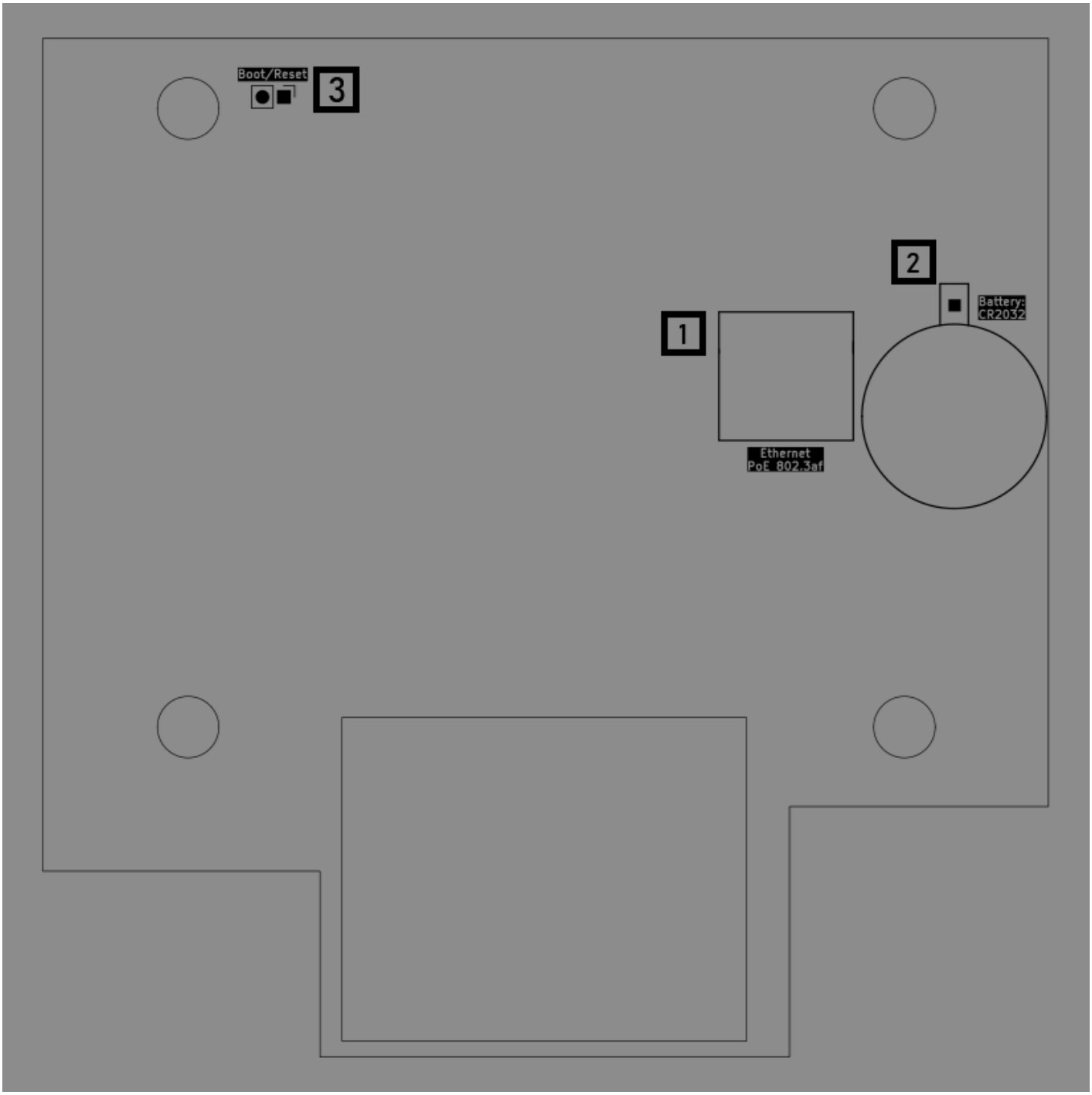
Because of different installation needs, the enclosure does not have any openings for wiring. Openings should be made by the customer anywhere in the rear section of the housing – this does not void warranty. Remember to use a cable gland with an IP rating of at least IP65 to maintain the declared enclosure ingress protection.

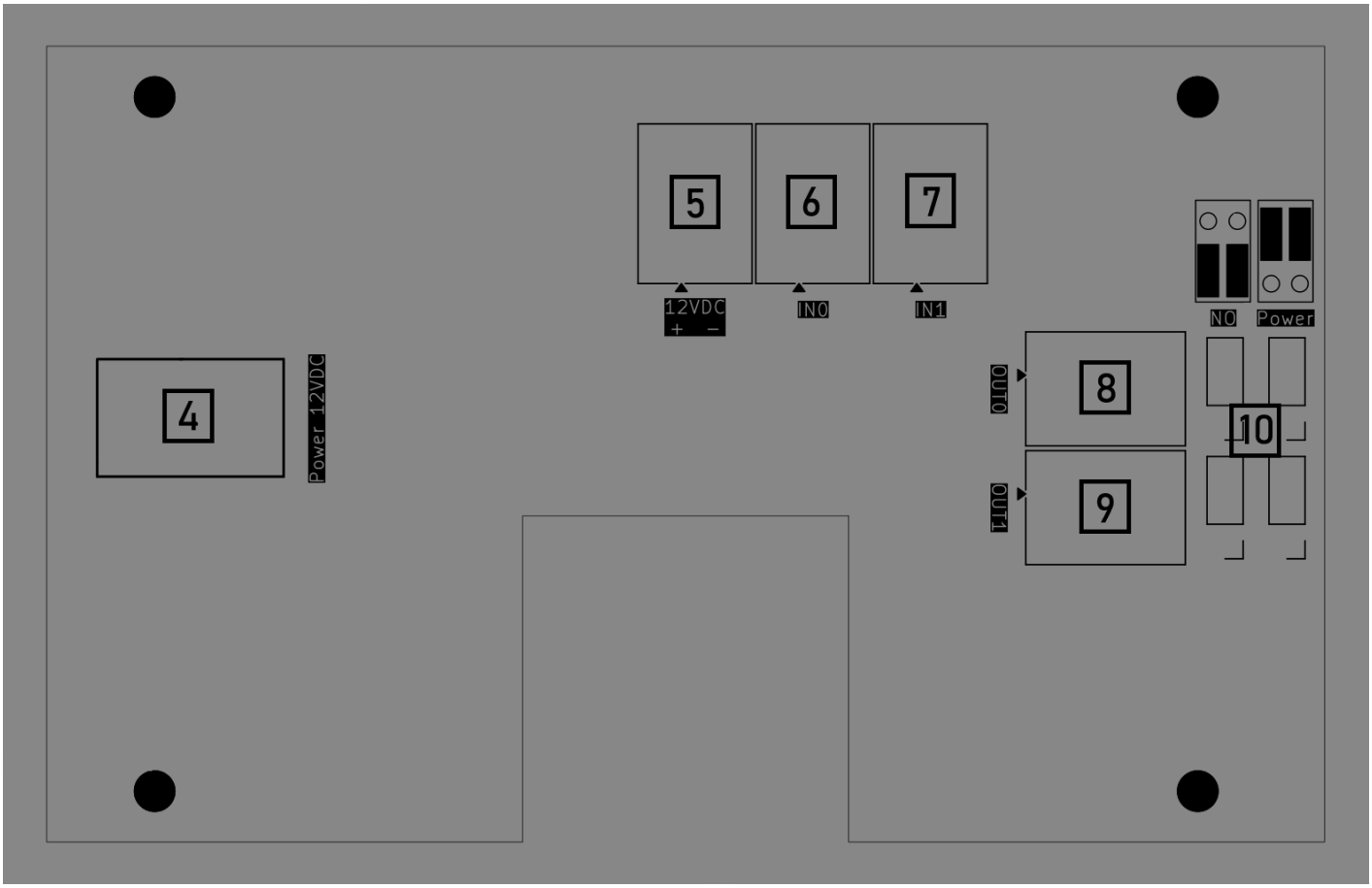
Dimensions



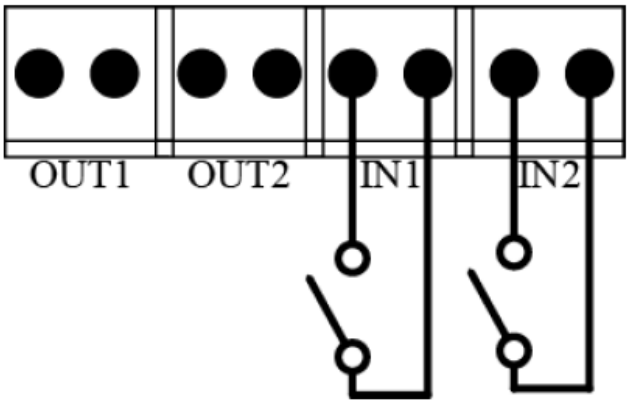


Connector description





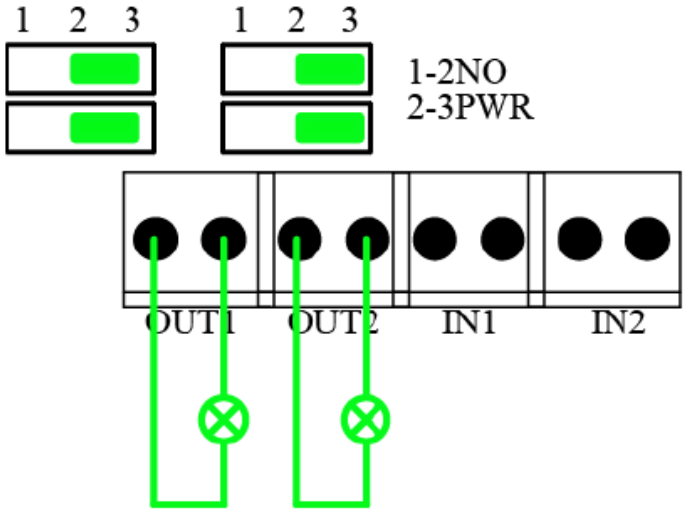
1. **LAN** – LAN connection socket,
2. **CR2032 battery holder** – For RTC backup,
3. **Reset** – **RESET** jumper. Shorting the pins for a time between 10-15 seconds restores the device to factory settings,
4. **Power 12VDC** – **12V DC** power input (5,5x2,1mm barrel jack),
5. **+12VDC-** – **12V DC** power input (quick-connect terminal),
6. **IN0** – General purpose input No. 0,
7. **IN1** – General purpose input No. 1,
8. **OUT0** – Relay No. 0 connecto,
9. **OUT1** – Relay No. 1 connecto,
10. **RelayMode** – Relay mode jumpers. In position 1-2 – NO relay output, 2-3 – PWR output – voltage is supplied from the **12V DC** power input.



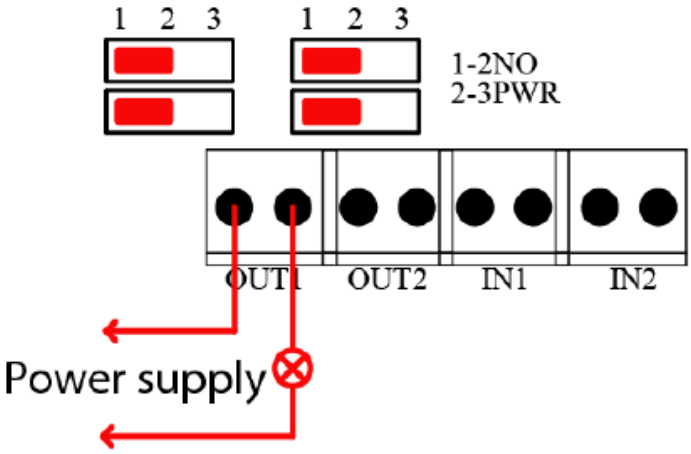
OUT1 and **OUT2** outputs can operate in two modes:

- **PWR mode** – Configuration jumpers set in position 2-3 (as shown in the image below). In this mode, an activated relay will provide the output with the power supply voltage eg. 12V DC. If the reader is powered with 24V DC, 24V DC will be provided.

> **Warning**
>
> **If the reader is powered with PoE 802.3af and does not have a supply connected to the power input, outputs OUT0 and OUT1 won't have any voltage supplied to them.**



- **Normally Open mode** – Jumpers set in position 1-2 (as shown in the image below). In this mode an external power source is necessary.



Communication channels and general characteristics

Proxer allows communication by **LAN** (WLAN) Urządzenie Proxer umożliwia komunikację przez sieć **LAN** (WLAN), offering a wide array of communication protocols which allow for remote status monitoring and device management:

- **Built-in WWW server**, accessible via a standard web browser,
- **HTTP/HTTPS** in server mode,
- **HTTP/HTTPS** in client mode,
- **Modbus TCP**,
- **MQTT/MQTTS** in client mode,
- **E-mail**.

> **Information**
>
> **The device supports advanced security options such as SSL encryption, HTTPS, certificate verification and an option to send encrypted e-mail messages.**

Device versions

Two Proxer versions are available:

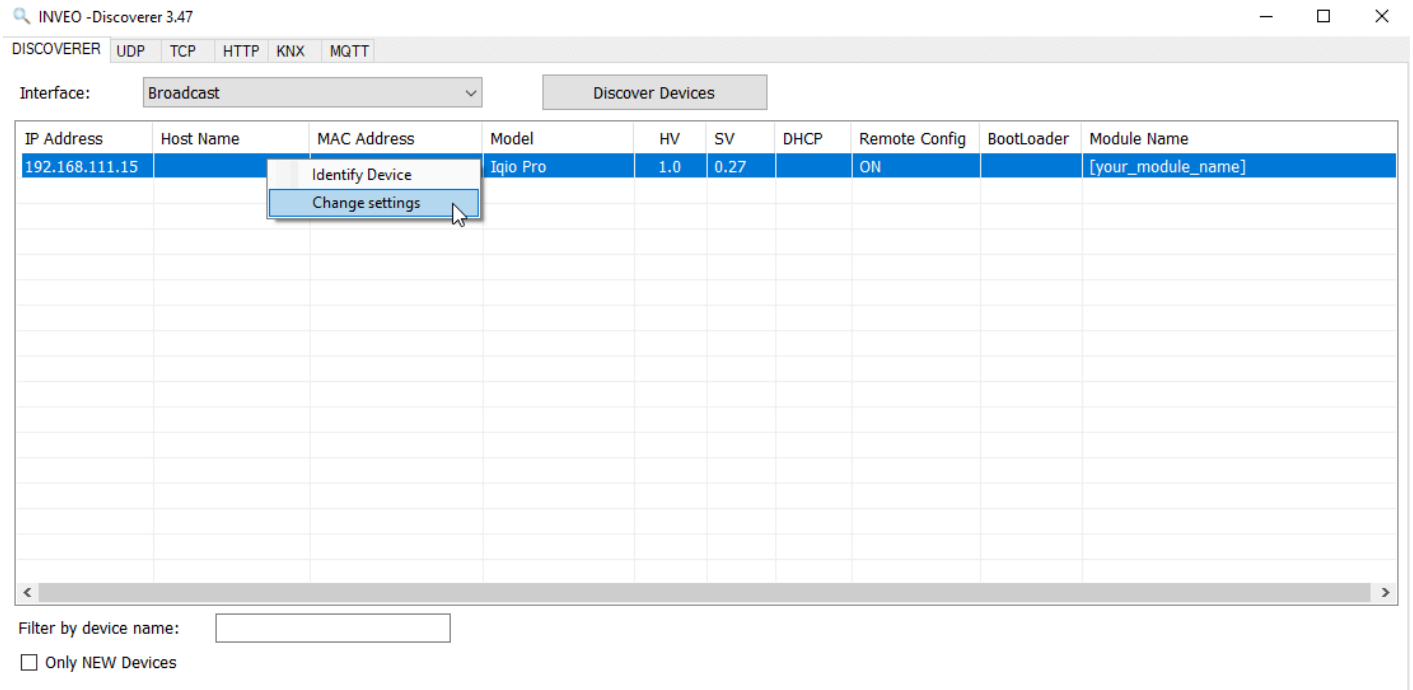
- **Standard model (LT)**,
- **Model with additional read field backlight nad adaptive brightness (SE)**.

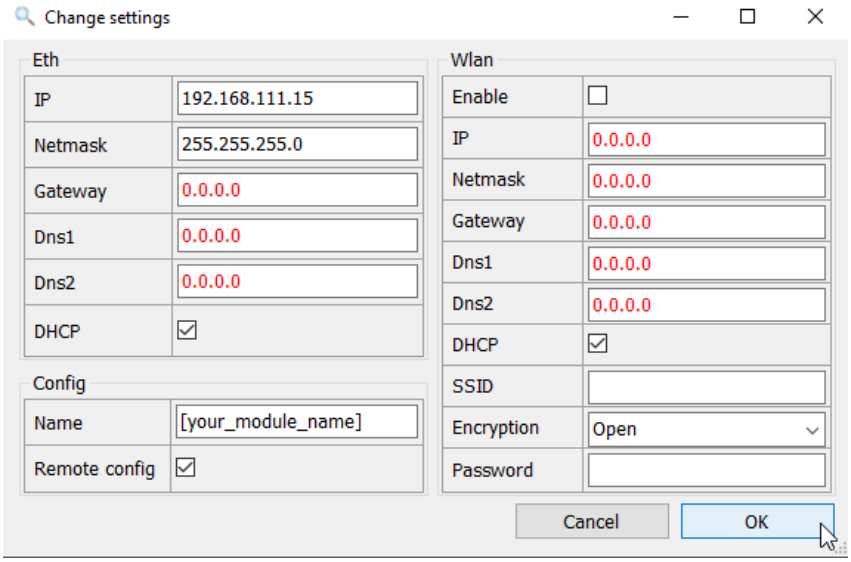
Network configuration

> **Warning**
>
> **After the initial power-up, it's needed to configure the device. It can be done in two ways, the simplest method is to use the Inveo Discoverer programme.**

Changing basic network parameters – Discoverer programme

1. Download and install the **Discoverer** programme.
2. After installing and starting the application:
   - Select the appropriate network interface from the list of available options. (It's best to use **Broadcast**),
   - Click **Discover Devices** to browse for all Inveo devices available on the network.
3. After the device list is displayed, right-click on the selected device and open the network configuration window.





Warning

If Remote Config is disabled (by default), you need to configure the device with the built in WWW server. This requires changing the subnet address of the computer.

Tip

Remote Config setting can be changed in the Administration tab of the WWW server. All you need to do is to select/deselect Enable remote config. After making the changes, it's required to apply them using Save.



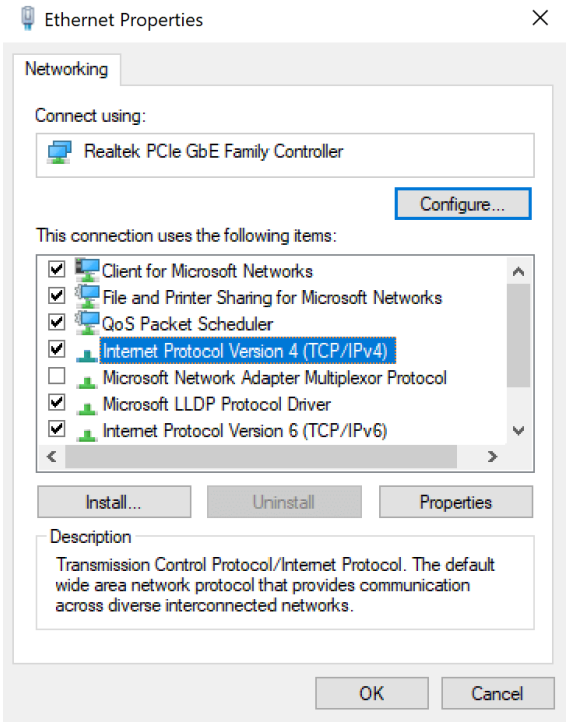
Changing the computer subnet address for configuration

During configuration without **Discoverer** programme, it's needed to change the subnet address of the computer on the same network.

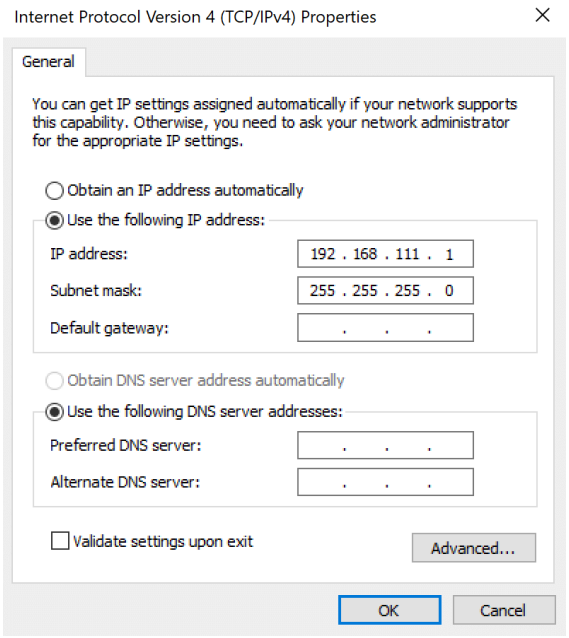
In order to access the computers network configuration, do one of the following:

- Press **Win + R**, type in `ncpa.cpl`, and then press **Enter**.
- Alternatively, go to:
  Start → Control Panel → Network and Internet → Network and sharing center → Change adapter settings.

Then:
1. Select the network connection.
2. Right-click it and select **Properties**.
3. After selecting this option, network connection configuration screen will appear.



Select **Internet protocol version 4 (TCP/IPv4)** and type in the following parameters:



Note

After applying the configuration with OK, start a web browser and type in the following address: 192.168.111.15. (Default user and password: admin/admin)

Configuring LAN and Wi-Fi

In the **Administration / Network** tab, it's possible to change the LAN (and WLAN - if available) parameters.

Name	Value	Description
DHCP		Enable Ethernet DHCP
IP	192.168.111.15	A.B.C.D
Netmask	255.255.255.0	A.B.C.D
Gateway	0.0.0.0	A.B.C.D
DNS1	0.0.0.0	A.B.C.D
DNS2	0.0.0.0	A.B.C.D

WLAN network configuration 

Name	Value	Description
Wi-Fi		Enable Wi-Fi
DHCP		Enable Wi-Fi DHCP
IP	192.168.111.15	A.B.C.D
Netmask	255.255.255.0	A.B.C.D
Gateway	0.0.0.0	A.B.C.D
DNS1	0.0.0.0	A.B.C.D
DNS2	0.0.0.0	A.B.C.D
Encryption	Open	Select Wi-Fi encryption
SSID		Wi-Fi SSID
Password		Wi-Fi Password

If **DHCP** is selected, the module will attempt to retrieve the IP address, subnet mask, network gateway, and DNS addresses from the DHCP server. In this case, **IP Address, IP Mask, Gateway, DNS1, DNS2** fields remain inactive.

If the DHCP server is not available, the device will set its own parameter values in the grayed-out fields.

The following fields are used for network configuration:

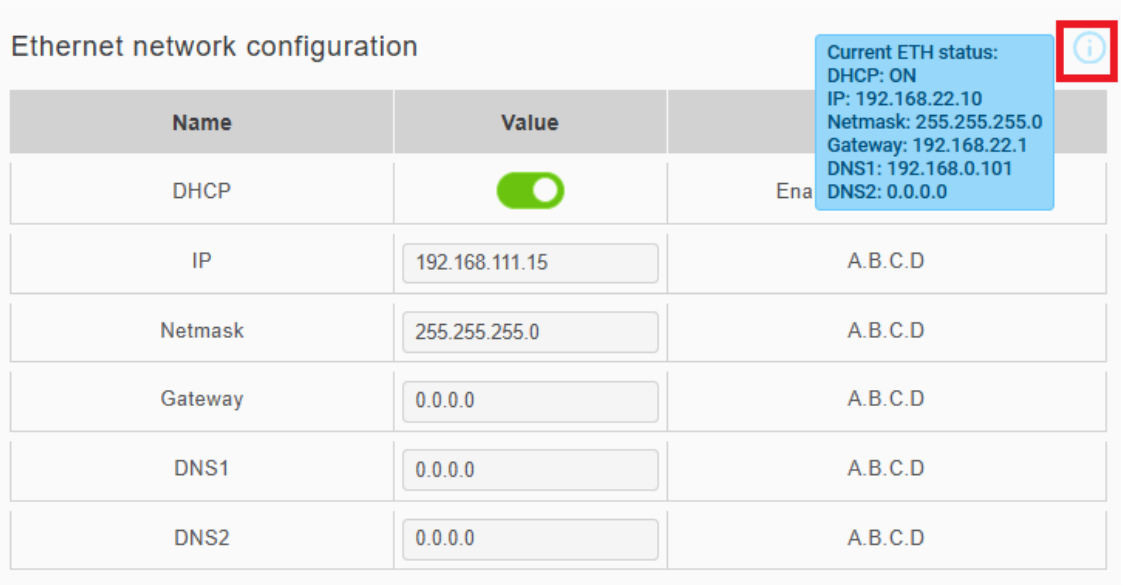
- **DHCP** – Enable/disable automatic retrieval of the IP address,
- **IP Address** – Module IP address (set manually if DHCP is absent),
- **IP Mask** – IP subnet mask,
- **Gateway** – Network gateway,

- **DNS1, DNS2** – DNS addresses,
- **SSID** – Wireless network broadcast name,
- **Password** – Wireless network password,
- **Encryption** – Encryption type.

> **Tip**
>
> **After making changes, apply them using the Save button**

Checking the network status

> **Information**
>
> **The device allows preview of current LAN and WLAN network parameters. In the Administration / Network tab, hover over the blue "i" icon. If DHCP is used, the information icon will display data retrieved from it.**

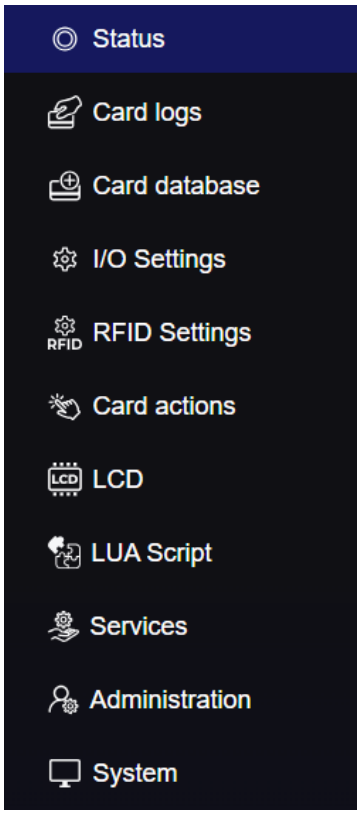


This section contains the following information:

- **IP** – Current IP address,
- **Mask** – Network mask,
- **Gateway** – Network gateway,
- **DNS1, DNS2** – DNS server addresses,
- **SSID** – Broadcast name of the WLAN, to which the device is connected.
- **RSSI** – Wi-Fi signal in dBm. Value of -110 indicates a no signal condition.

Device configuration

The device's home page accessible via a web browser is the device's main user interface that allows for quick access to important information about the current device condition. By default it's accessible by IP address 192.168.111.15, but the user has the ability to change this address in device's network settings.



An informational ribbon is located in the upper portion of the website, informing the user about key device data, such as: model, IP address, unique user-assigned name, firmware version, and MAC address.



Device status

Detailed information about the input and output states, as well as operational parameters of the device are displayed on the home screen.

Status

Name	Value
Input #0	0
Input #1	0
Output #0	0
Output #1	0
Last read UID:	
Number of read UIDs:	0
Last card action:	Action on-server
Uptime:	03:21:41

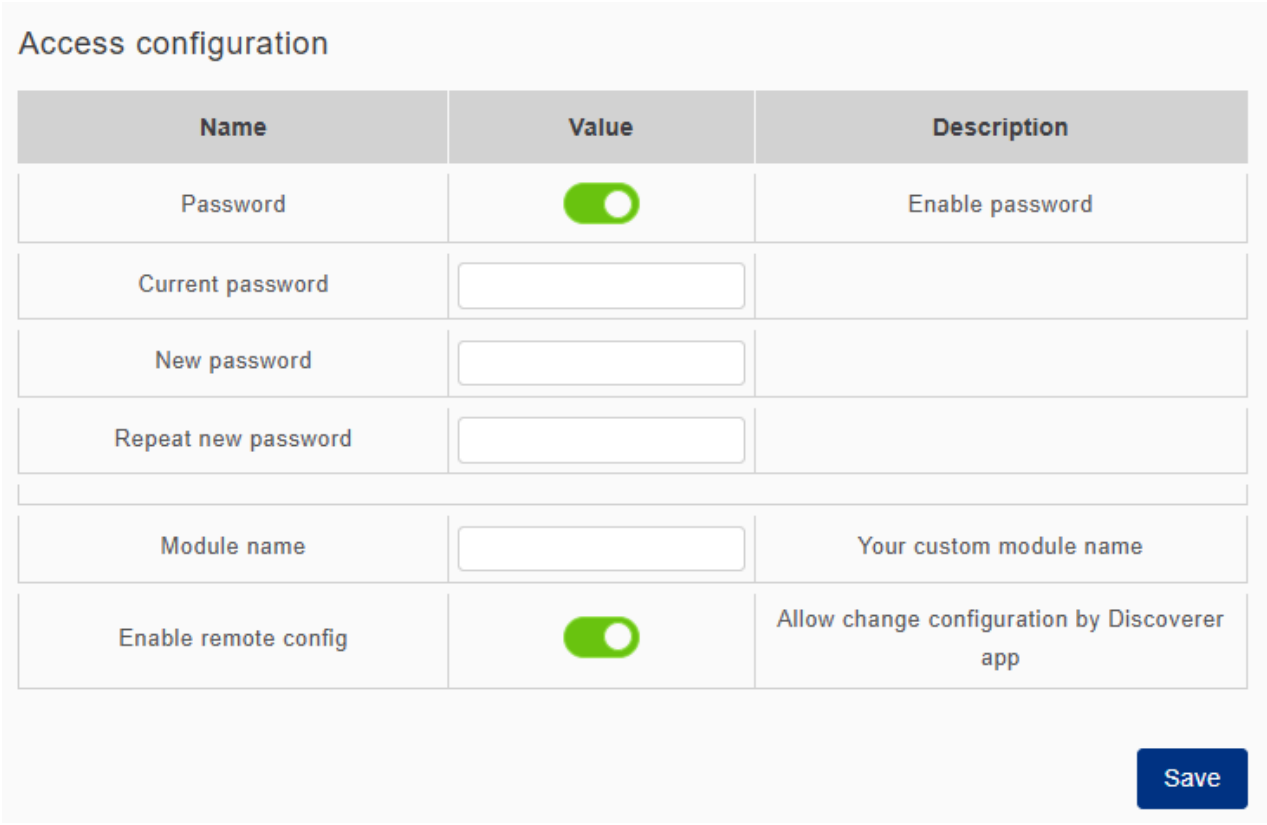
- **Input #0** – Displays the current state of the first binary input,
- **Input #1** – Displays the current state of the second binary input,
- **Output #0** – Informs about the current state of the relay output No. 0,
- **Output #1** – Informs about the current state of the relay output No. 1,
- **Last read UID** – Displays the ID of last read RFID tag in hexadecimal format (HEX),
- **Number of read UIDs** – Displays the number of RFID tags read since last device restart,
- **Last card action**,
- **Uptime** – Time since last device restart.

Administration

The **Administration** tab in device's user interface enables management of key system settings.

Access

In the **Access** section located in the **Administration** tab, the user can manage the external device access. This applies to authentication, module name, Discoverer software and protocol access.



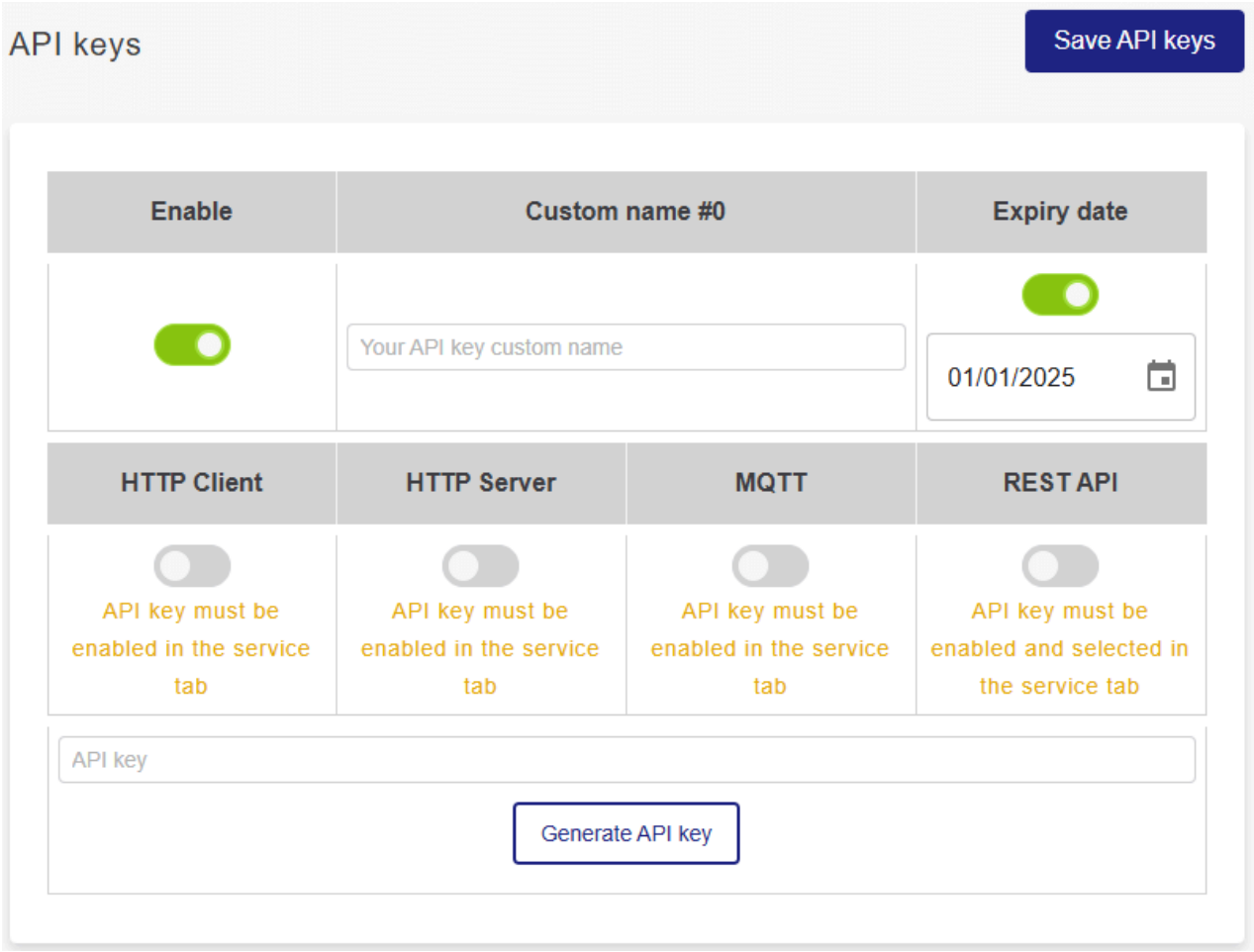
- **Password** – Enable/disable configuration page password requirement,
- **Current password**,
- **New password**,
- **Repeat new password**,
- **Module name** – Displayed in the Discoverer software. Assigning a unique name makes identification of devices within a system easier,
- **Enable remote config** – Enable/disable Discoverer software access.

Information

Default device access settings:
- login: admin
- password: admin

API KEYS

The device allows the user to configure a set of five unique API keys, used for authentication and identification of users or processes that initiate communications with the device.



- **Enable** – Enable/disable the selected API key,
- **Custom name** – Key name assigned by the user,
- **Expiry date** – API key expiry date,
- **HTTP Client, HTTP Server, MQTT, REST API** – Selection of the communication protocol that will use the specified API key,
- **API key** – Key value entry field,
- **Generate API key** – This button generates a random API key that will automatically be inputted into the **API key** field.

Tip

Changes made in the API keys tab should be saved with Save API keys..

When sending **JSON** instructions to the reader, the API key (example value) should be entered as follows:

```
    "apikey":"4be38890-4749-456a-b427-5a224d4dd890",
    "out_run":0,
    "buzz":1
```

If **RAW** data will be sent, it should follow this example:

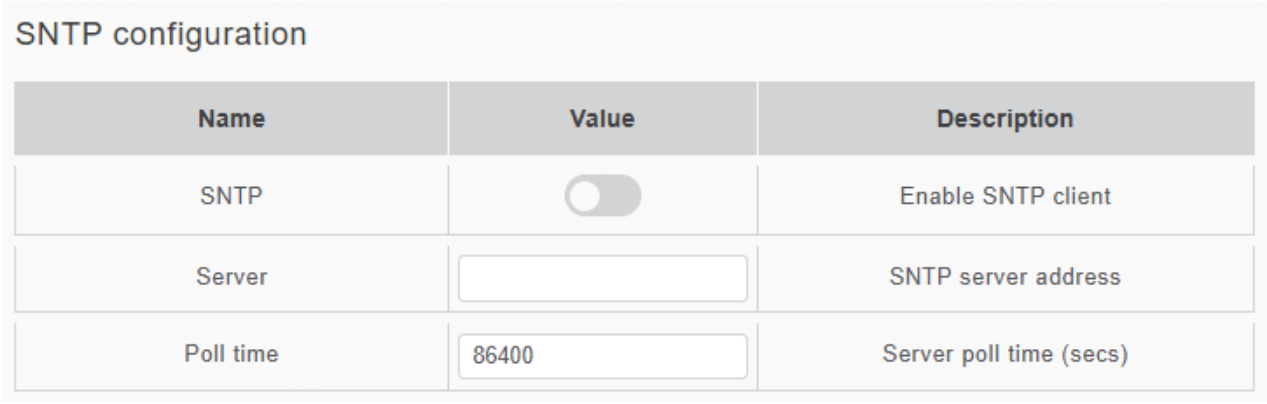
```
apikey=4be38890-4749-456a-b427-5a224d4dd890&out_run=0&buzz=1
```

Time settings

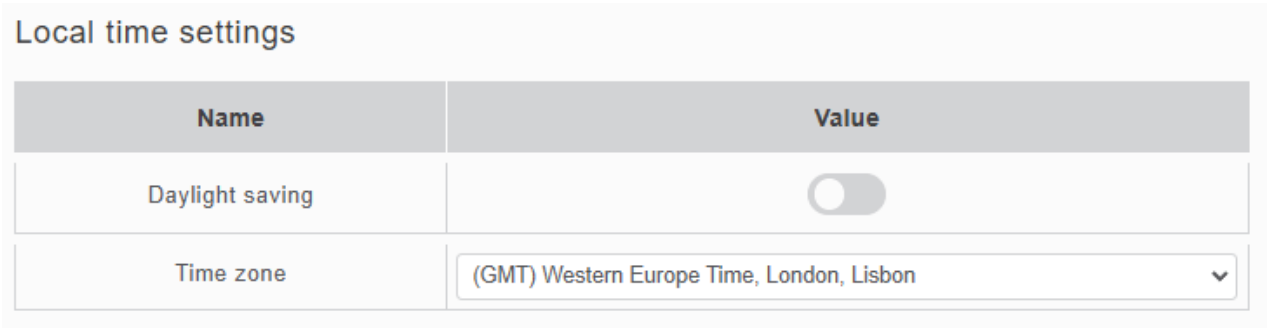
In the **Administration / Time** section, date and time settings are located. The device is equipped with a Real Time Clock with a battery backup, which allows for correct time measurement even during a power loss.

In the **SNTP configuration** table, a time server for downloading current time can be specified:

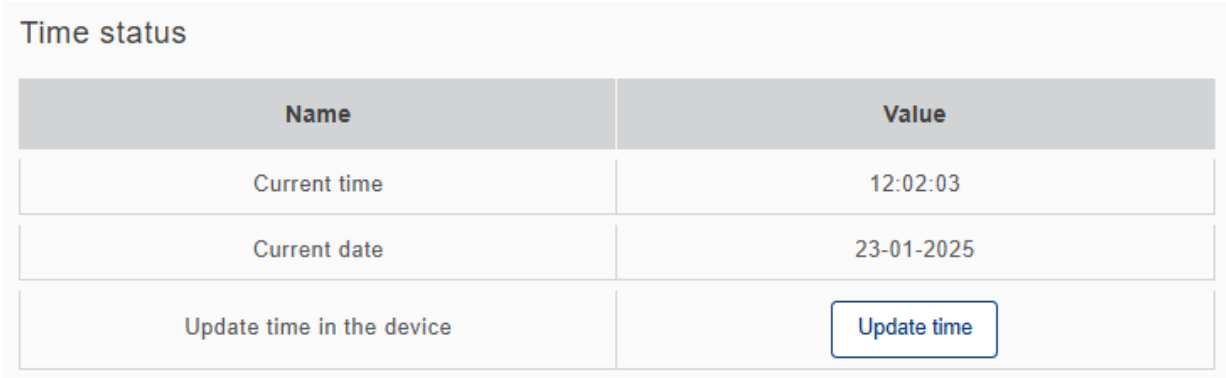
- **SNTP** – Enable the SNTP client,
- **Server** – IP address or domain name of the SNTP server,
- **Poll time** – Time server querying frequency.



The **Daylight saving** option allows for automatic switching between daylight savings time. **Time zone** allows selection of a time zone.

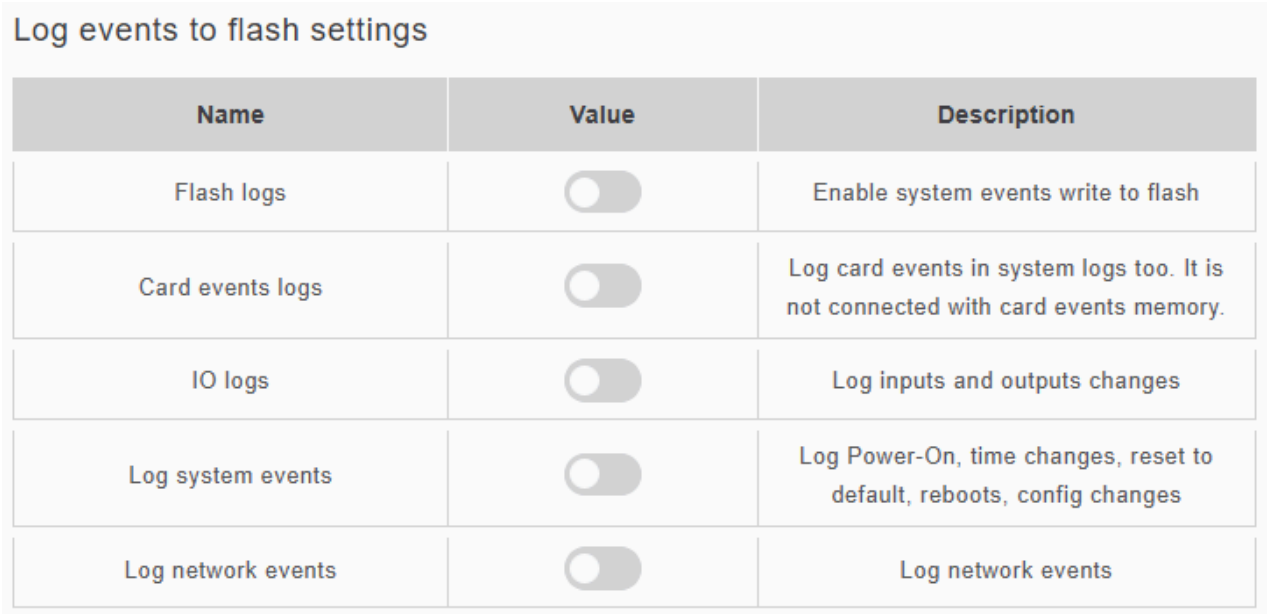


The **Update Time** button synchronizes the device's internal clock with the current PC time.



Diagnostics, events

The module offers advanced diagnostic options, allowing for data logging and sending them to a Syslog server. The settings are located in the **Administration / System Events** tab.



The **Log Events to flash settings** table allows the user to configure events, that are to be logged to flash memory:

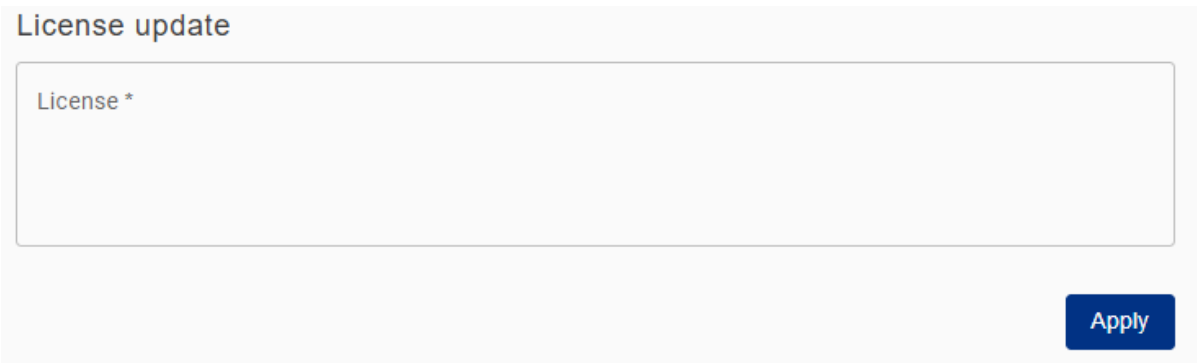
- **Flash logs enable** – Enable logging events to flash memory,
- **Cards event logs** – Log card events - independent of card memory,
- **IO logs** – Log input and output state changes,
- **Log system events** – Log system events eg. module shutdown, changes in time settings, reset, restart and configuration changes,
- **Log network events** – Log network events eg. establishing a Wi-Fi connection.

Information

All data can be viewed in the System/Internal logs tab

Adding licenses

The **License update** field in the **Update** tab is used to add new or update the current licenses.



I/O Settings

The **I/O Settings** tab enables configuration of input and output related settings. The reader is equipped with two relay outputs and two dry contact digital inputs. The user can adjust relay operations and input configuration in accordance to application needs.

Relay outputs configuration:

Name	Value	Description
Out #0 Mode	On period	Output #0 action
Out #0 Time	40	Time [0.1 s]
Out #0 Time Long	100	Time [0.1 s]
Out #0 Invert	None	Invert output

- **Out mode** – Relay output operation mode:
  - **None** – Software disabled relay control. The relay is inactive,
  - **Off** – Permanent relay deactivation,
  - **On** – Permanent relay activation,
  - **Toggle** – Change of relay state to the opposite (activation if it was deactivated, or deactivation if activated),
  - **On period** – Relay activation for a time defined in the **Out Time** field,
  - **On long period** – Relay activation for a time defined in the **Out Time Long** field.
- **Out Time** – Time of relay activation for the **On Period** mode. The user can enter a value in 0,1 second units, during which the relay will remain active,
- **Out Time Long** – Time of relay activation for the **On long period** mode. The user can enter a value in 0,1 second units, during which the relay will remain active,
- **Out Invert** – Software change of relay state - default NO (Normally Open); selecting this option changes it to NC (Normally Closed).

Digital input settings:

Input #0	None	Input action
Input #0 invert	None	Invert input
Input #0 filter	0	Input transient filter [ms], set 0 to disable

- **Input** – Defines the action that will be undertaken when the input is activated:
  - **None** – No action,
  - **Trigger output #0** – Trigger the relay output No. 0,
  - **Trigger output #1** – Trigger the relay output No. 1.

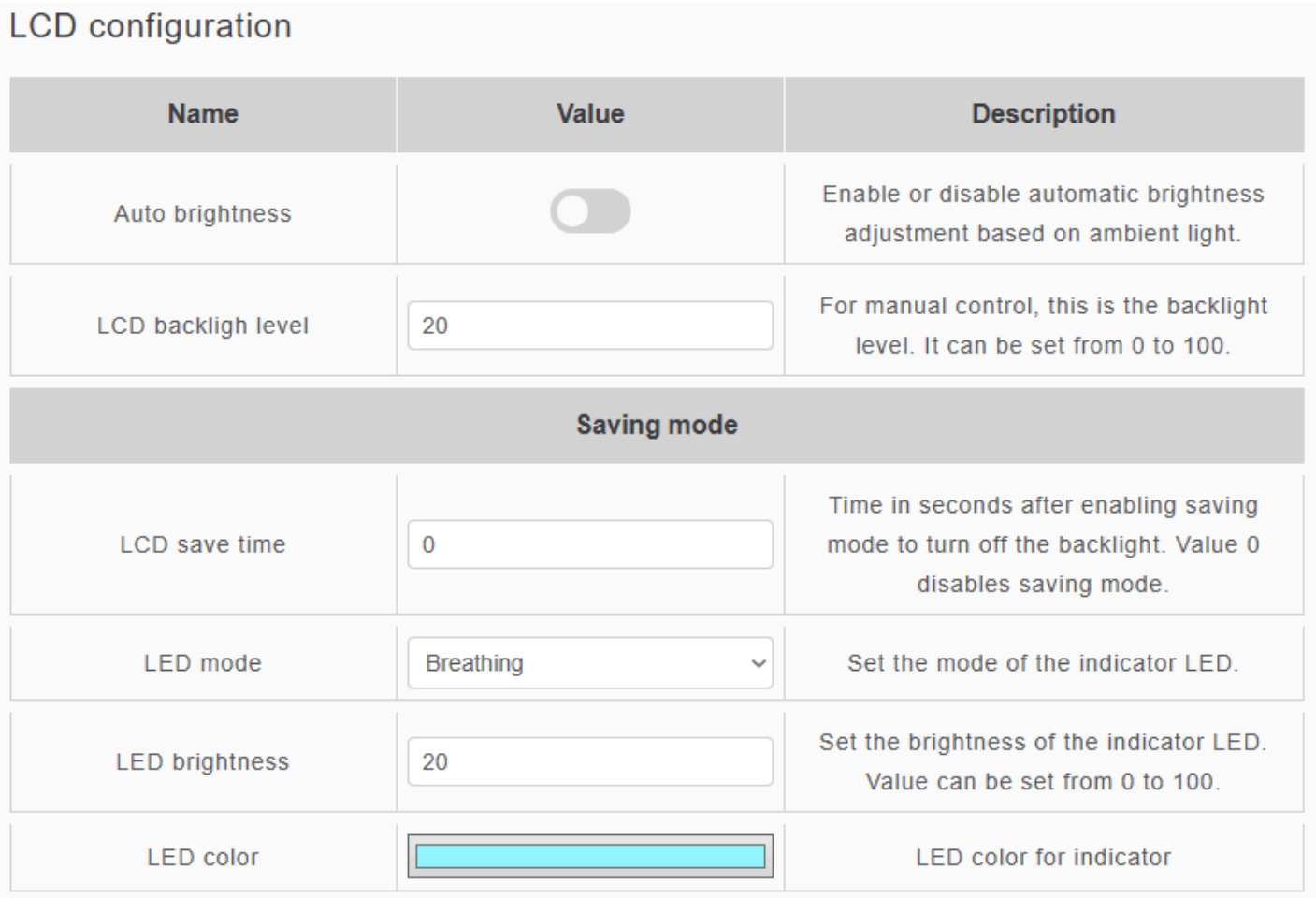
- **Input invert** – Software inversion of the default input state,
- **Input filter** – Allows to define a time (in ms) during which the device ignores short changes to input signal, treating them as interference. The value of 0 disables filtration, introducing no delays.

LCD

Both Proxer versions are equipped with colour LCD display that can be used to display animations, graphics and text configured by the user.

Settings

The **Settings** allows configuration of global display settings.



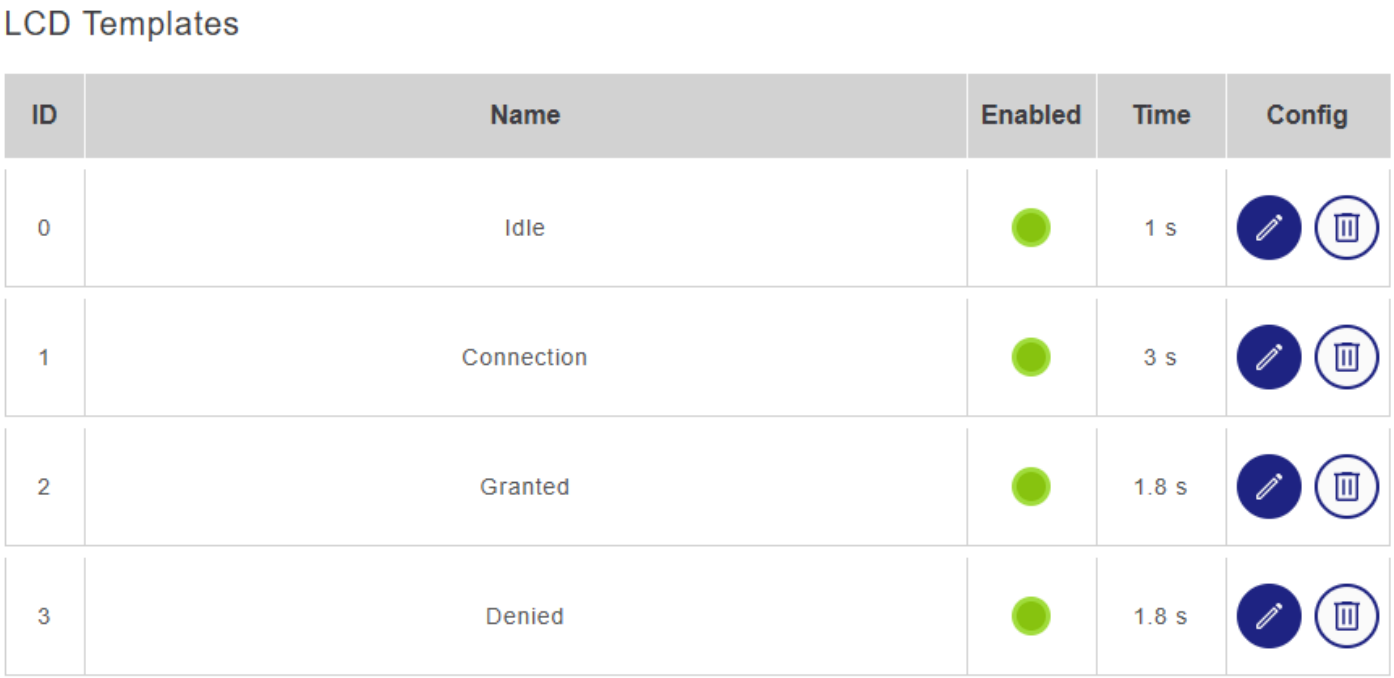
- **Auto brightness** – Enable/disable automatic LCD backlight and read field LED ring *(Proxer SE only)* brightness control,
- **LCD backlight level** – Display brightness in manual control mode,
- **LCD save time** – Time, after which the display will automatically go to sleep. Entering **0** will make the display work indefinetly,
- **LED mode** – Read field LED ring operation mode during sleep state *(Proxer SE only)*:
  - **Unchanged**,
  - **Off**,
  - **On**,
  - **Blinking**,

- **Heartbeat**,
- **Pulsing**,
- **Breathing**,
- **Cycle**.

- **LED brightness** – LED brightness in manual control mode,
- **LED color** – Read field LED ring backlight color during sleep state.

Templates

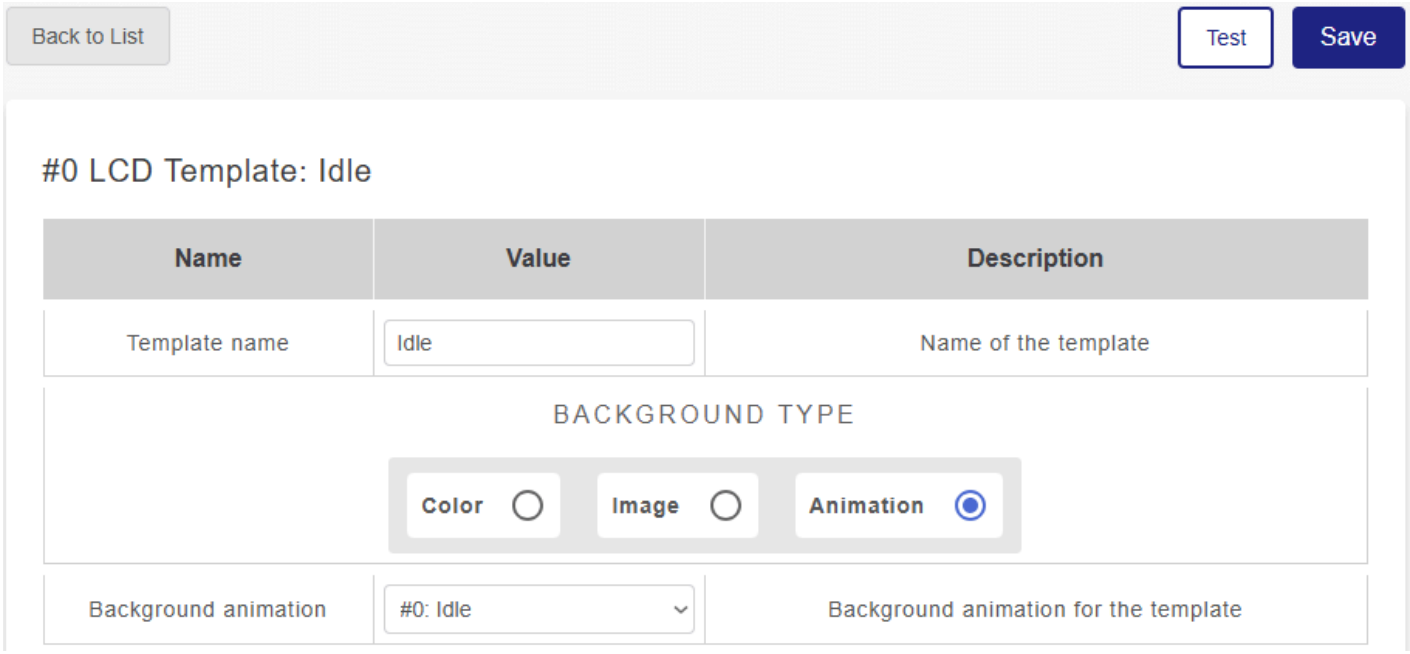
In this tab, the user can define and edit up to 10 templates of display and LED settings, for use during idle state or in response to card events.



Tip

**The Test button at the top of the page allows for instant preview of configured settings directly on the reader.

Edit view



- **Template name** – Name of the template used for easy identification,
- **Background type**:
  - **Color** – Still color that can be selected in **Background color** below,
  - **Image**,
  - **Animation**.

> **Tip**
>
> **The process of uploading animations and graphics has been described in detail in the Inveo Assets Builder program user manual.**

LED configuration – Proxer SE only

LED Configuration		
LED mode	Breathing	Set the mode of the indicator LED.
LED brightness	20	Set the brightness of the indicator LED. Value can be set from 0 to 100.
LED color		LED color for indicator

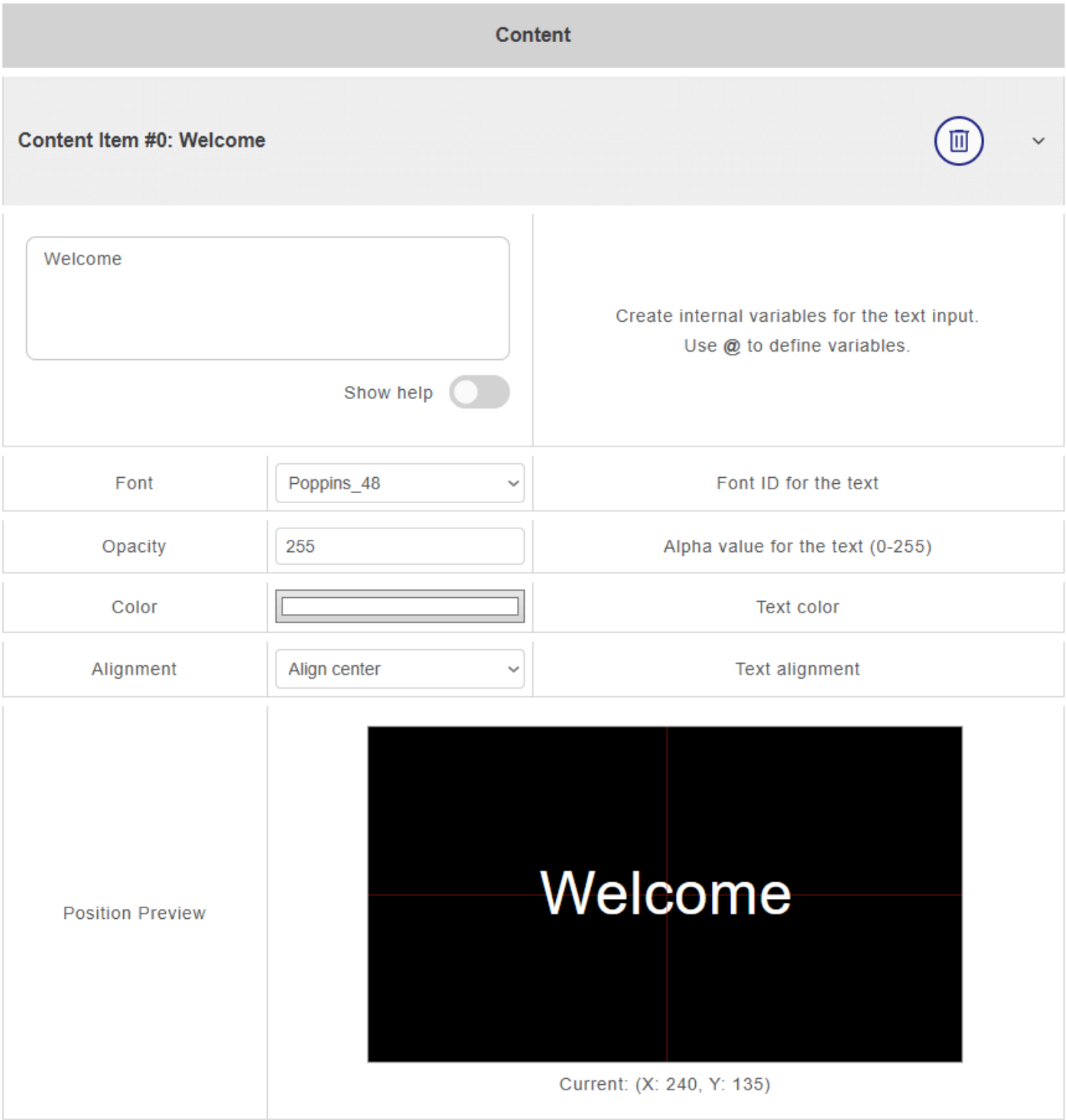
- **LED mode**:
  - **Unchanged**,
  - **Off**,
  - **On**,
  - **Blinking**,

- **Heartbeat**,
- **Pulsing**,
- **Breathing**,
- **Cycle**.

- **LED brightness** – LED brightness in manual control mode,
- **LED color** – Read field LED ring backlight color.

CONTENT

The user can define up to 4 text content blocks per template. They will be displayed in front of the specified background.



- Text field. **Predefined variables** can be used,
- **Font**,
- **Opacity**,
- **Color** – Text color,
- **Alignment**:
  - **Align left**,
  - **Align center**,
  - **Align right**.
- **Position preview** – Select and preview text position.

> **Tip**
>
> **After making changes, save them using the Save button**

Example: displaying text via HTTP Client

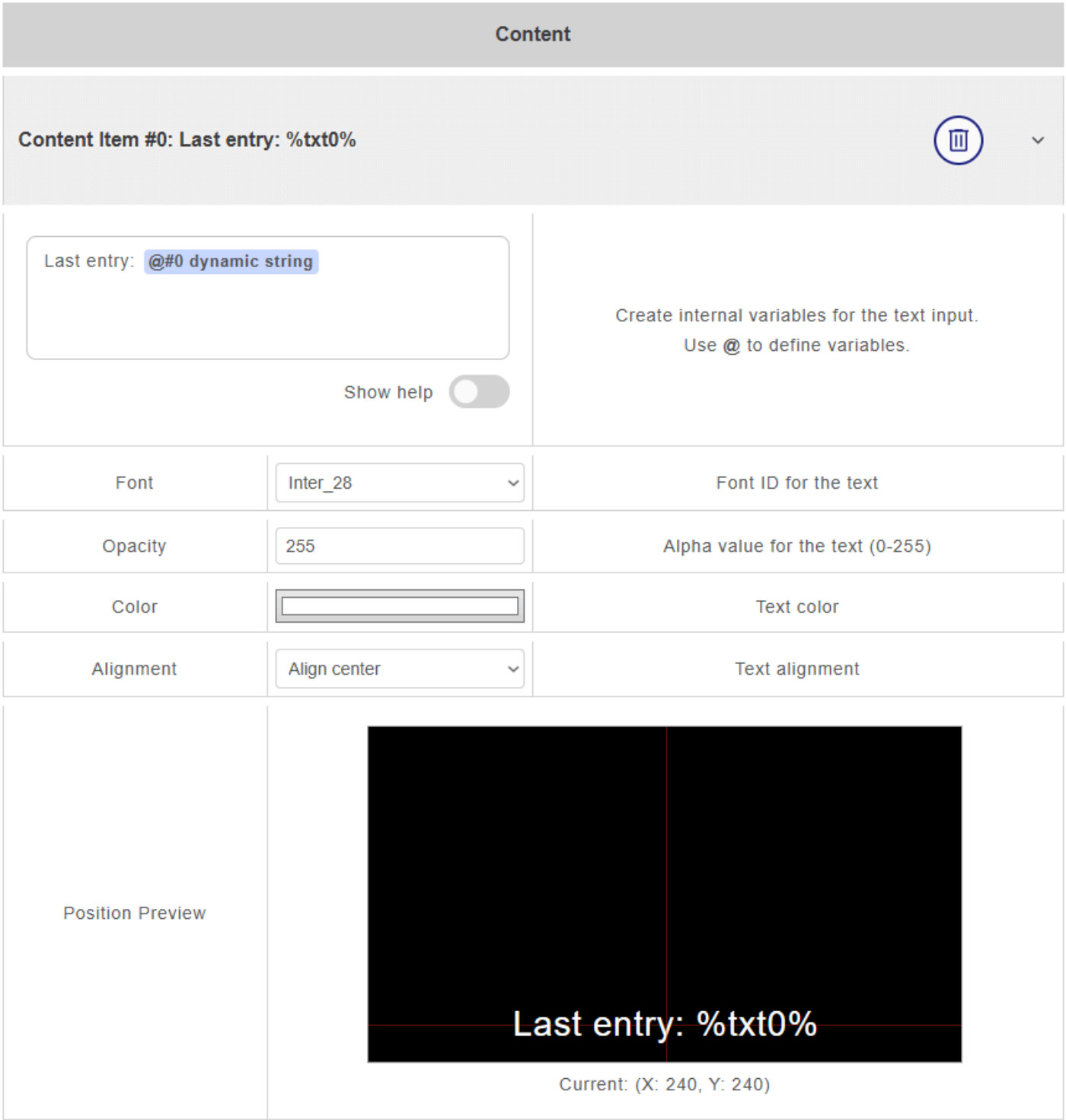
Unlike other readers equipped with 4x20 character monochrome LCD's, Proxer gives the user 4 `txt` variables that can be placed anywhere on the display.
Before any additional information from the server can be displayed, the variables must be added and placed on the desired template.

In this example, using HTTP Client, after a card is found in the server's database, template No. 3 (**Granted**) should be displayed, containing additional: **Last entry: [time of last entry]**, with the time changing dynamically depending on the last entry event.

LCD > Templates > Granted

#2 LCD Template: Granted

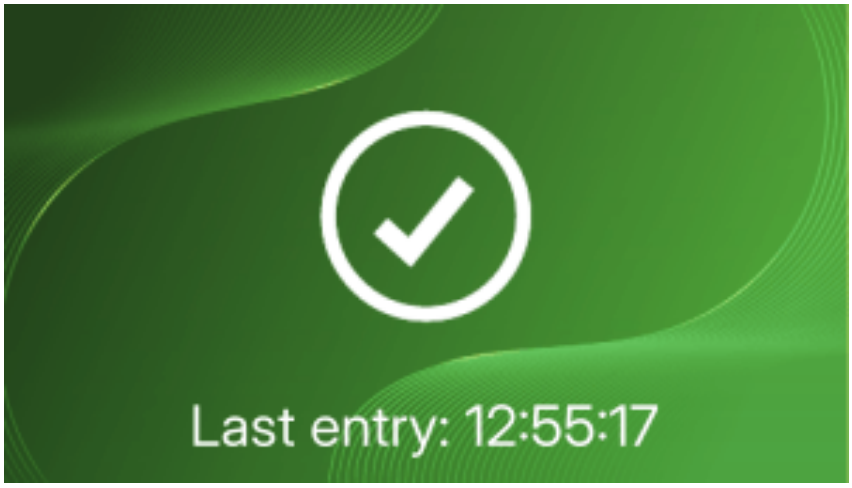
Name	Value	Description
Enable		Enable or disable this template
Template name	Granted	Name of the template
BACKGROUND TYPE Color ○ Image ◉ Animation ○		
Background image	#2: Granted	Background image for the template
Display time (ms)	5000	Time in milliseconds to display this template before switching to the next one
Which template to display next	#0 Idle	Template ID to display after this one ends (0 for idle)



After applying a known card, configured data will be sent (consult the **HTTP Client** chapter). In response, the server will send the following JSON file:

```
HTTP/1.1 200 OK
Content-Length: 57
Content-Type: application/json
{
"txt0":"12:55:17",
"template_name":"Granted",
"buzz":"1"
}
```

Result:


Last entry: 12:55:17

Predefined variables

The user has access to a set of predefined variables that can be used to configure display text, buttons *(Clocker CL-20, CL-30)* and other functions:

Variable	Description
`%time%`	Current device time
`%date%`	Current device date
`%timedate%`	Current device date and time (YYYY-MM-DD HH:MM:SS)
`%ts%`	Current device timestamp
`%mod_name%`	Device name
`%mod_model%`	Device model
`%eip%`	Ethernet – IP address
`%emac%`	Ethernet – MAC address
`%wip%`	Wi-Fi – IP address
`%wmac%`	Wi-Fi – MAC address
`%last_sntp%`	Timestamp of the last SNTP server time update
`%uid%`	Last UID number read (post conversion)
`%usr_name1%`	**Name #1** field from the card database (empty if no record)
`%usr_name2%`	**Name #2** field from the card database (empty if no record)
`%usr_id%`	ID from the card database (value `-1` if no record)
`%userdb_ver%`	Internal user database version
`%presence%`	Card presence in the read field
`%curid%`	UID of the card present in the read field (post conversion)

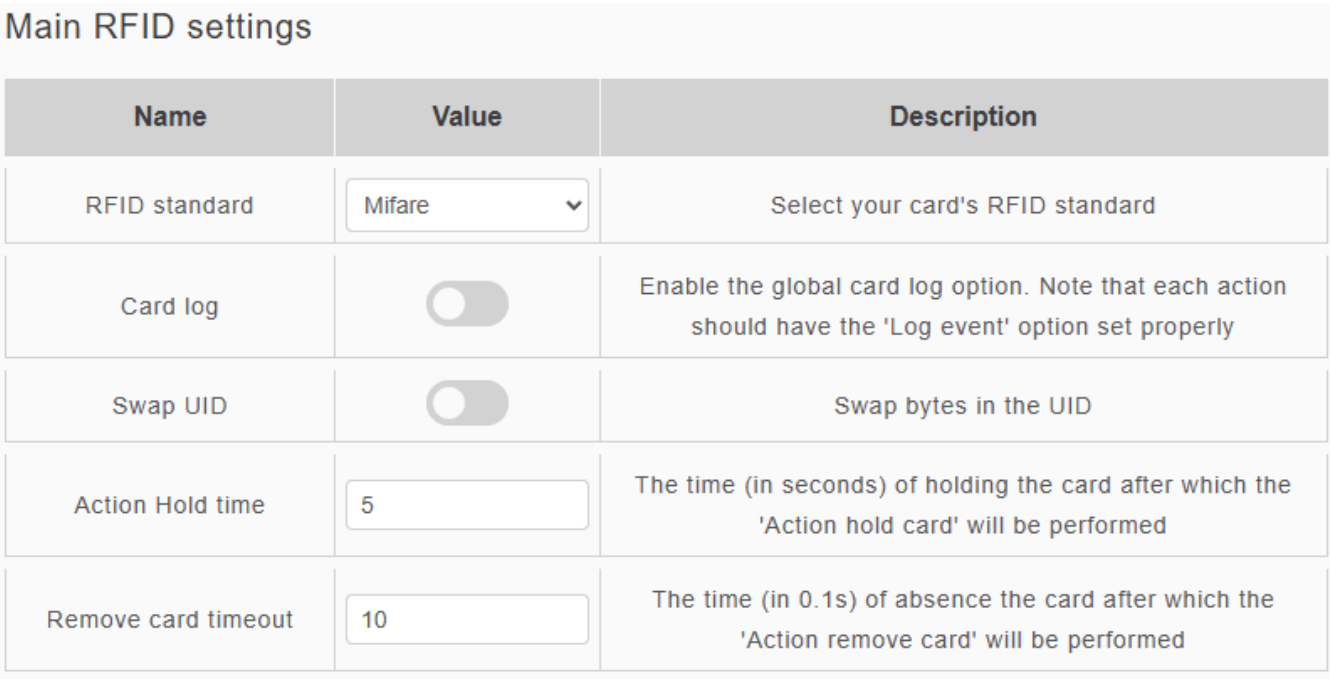
`%read_cnt%`	Amount of RFID tags read since the last device restart
`%txt0%`	Special purpose dynamic variable No. 0
`%txt1%`	Special purpose dynamic variable No. 1
`%txt2%`	Special purpose dynamic variable No. 2
`%txt3%`	Special purpose dynamic variable No. 3
`%enter_state%`	Function buttons state
`%enter_inout%`	Text displayed during entry/exit selection
`%enter_opt%`	Text displayed during optional button selection
`%btn%`	Last button pressed
`%json_cardlogs%`	24 oldest card log records
`%n_logs%`	Amount of logs in the reader's memory
`%ts_on%`	Timestamp of the On Card event
`%ts_hold%`	Timestamp of the Card Hold event
`%ts_remove%`	Timestamp of the Card Remove event
`%ts_last%`	Timestamp of the last card application
`%data%`	Contents of read data block if block reading function is enabled
`%fuid%`	Factory UID of the currently applied card
`%data_valid%`	Read data valid – returns 0 when a block read function encounters an error
`%in0%`	State of input No. 0
`%in1%`	State of input No. 1
`%cnt0%`	Counter value for input No. 0
`%cnt1%`	Counter value for input No. 1

`%out0%`	State of output No. 0
`%out1%`	State of output No. 1

RFID Settings

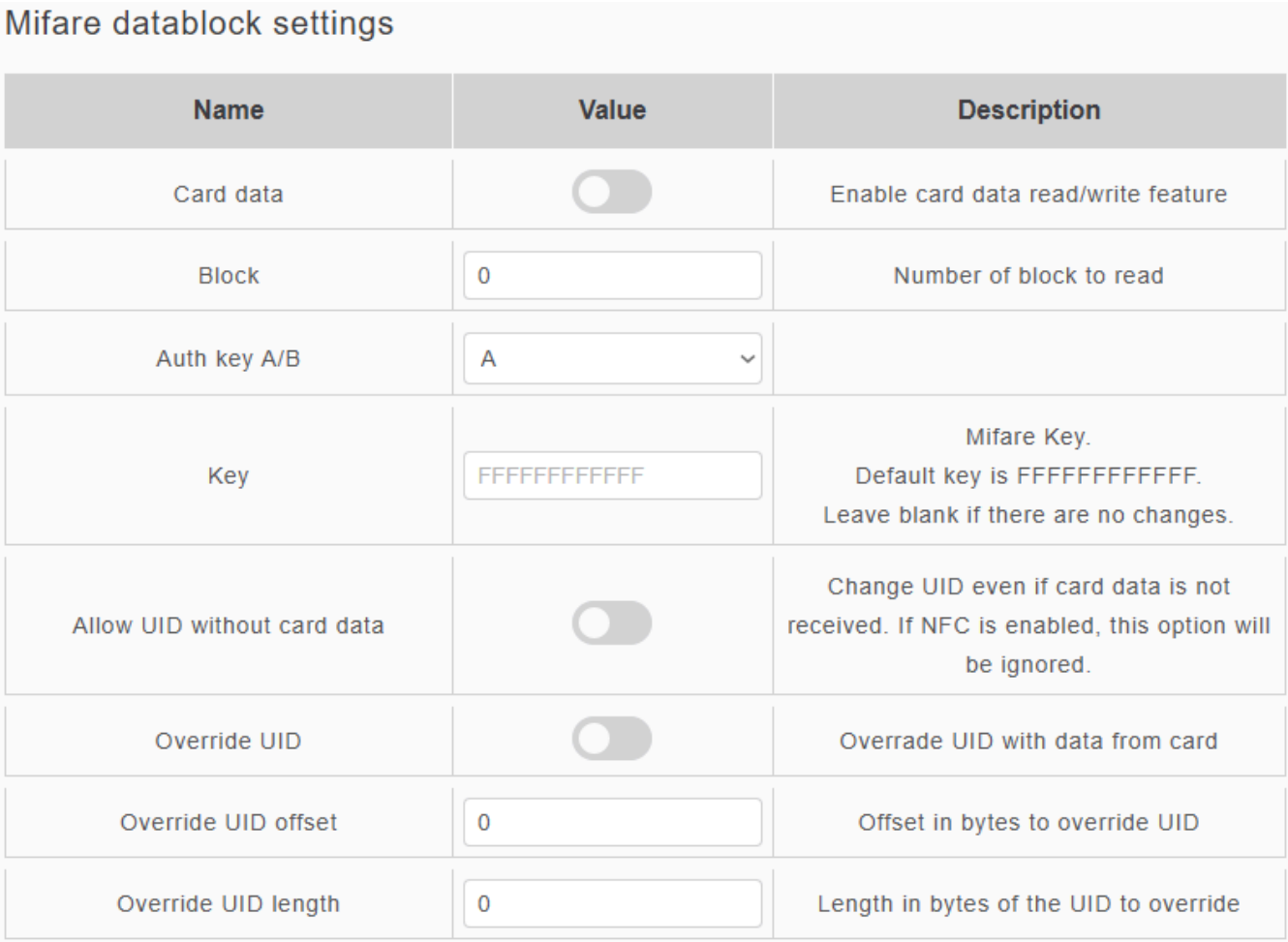
The tab allows the user to select an RFID standard and configure tag read, NFC and external reader parameters.

Setup – RFID standard selection



- **RFID standard**. For example, in the HF version, the following options are available:
  - **Mifare**
  - **ICODE**
  - **iClass – CSN only**
- **Card log** – Global control over card events logging,
- **Swap UID** – Swap UID bytes,
- **Action Hold time** – Amount of time (in seconds) a card needs to be present in the read field to trigger the **Action hold card**,
- **Remove card timeout** – Amount of time (in 0,1s units) of card absence to trigger the **Action remove card**.

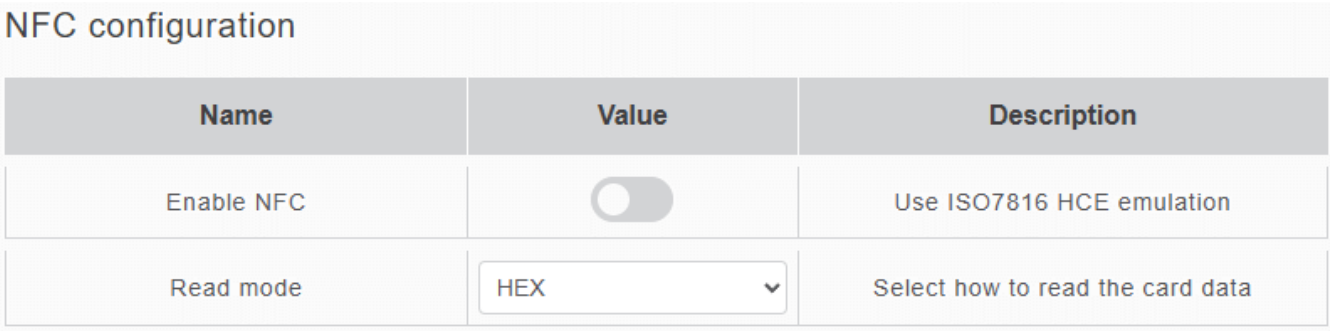
Card data – Only for HF version when Mifare is selected



- **Card data** – Enable/disable Mifare card data reading/writing function,
- **Block** – Read block address,
- **Auth key A/B** – Authentication key selection (default A),
- **Key** – Authentication key in HEX format - the key is saved in secure memory (write only). The key should be written only when a change is required. Default key: 6xFF (FFFFFFFFFFFF),
- **Allow UID without card data** – Overwrite card data without reading it previously (always enabled with NFC),
- **Override UID** – Override card ID with read data,
- **Override UID offset** – Character that the data readout begins from,
- **Override UID len** – Length of the data string to be read.

NFC – Only for HF version when Mifare is selected

The reader has the option to assign a smartphone with the **Inveo NFC iTag** application installed as an RFID tag, allowing the user to e.g. access the room without the need to physically posess additional keycards or RFID tags.



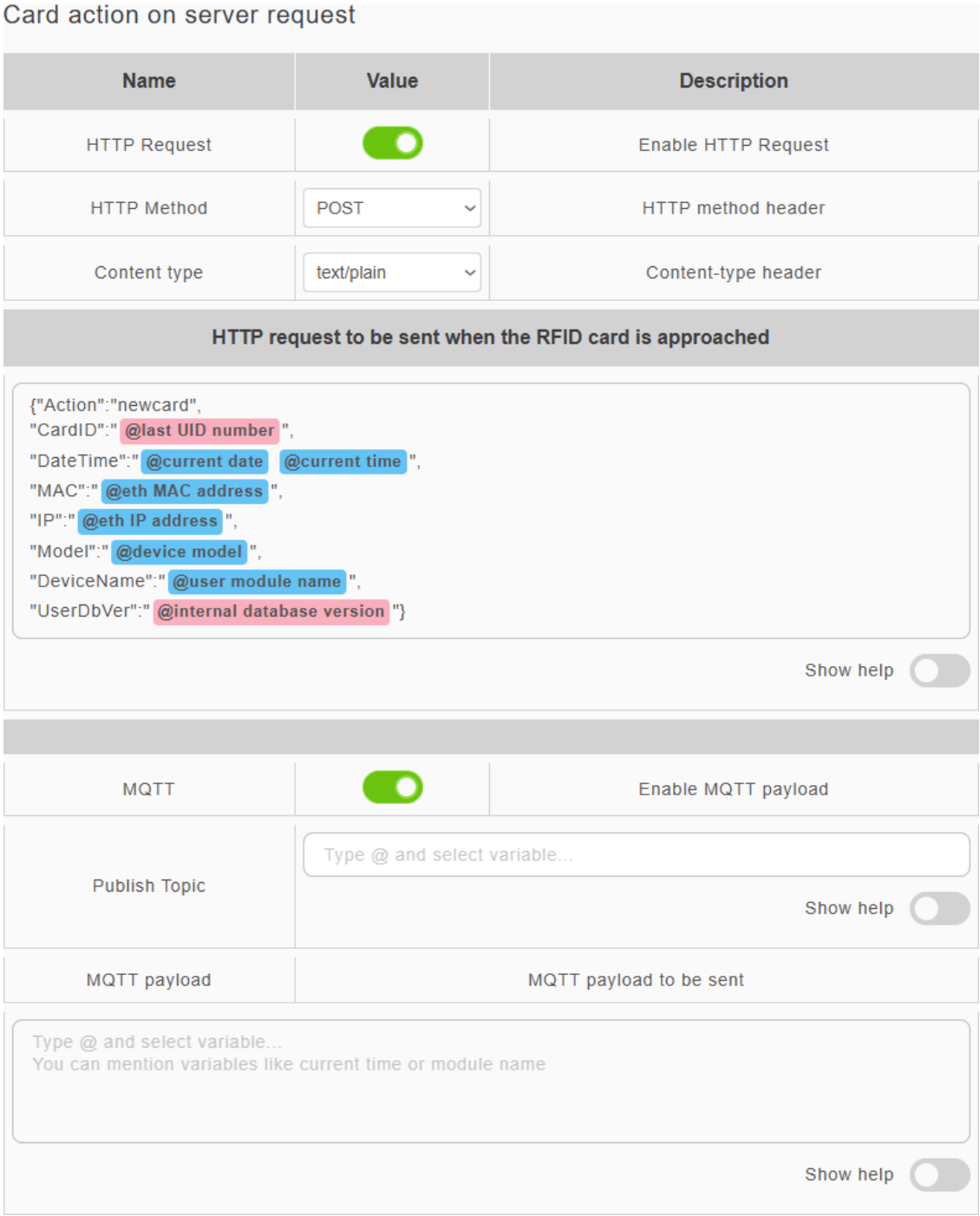
- **Enable NFC** – Enable the smarphone-emulated card reading function,
- **Read mode** – Data reading mode selection:
  - **HEX** – For Android client devices,
  - **Plain text** – For Android and iOS devices.

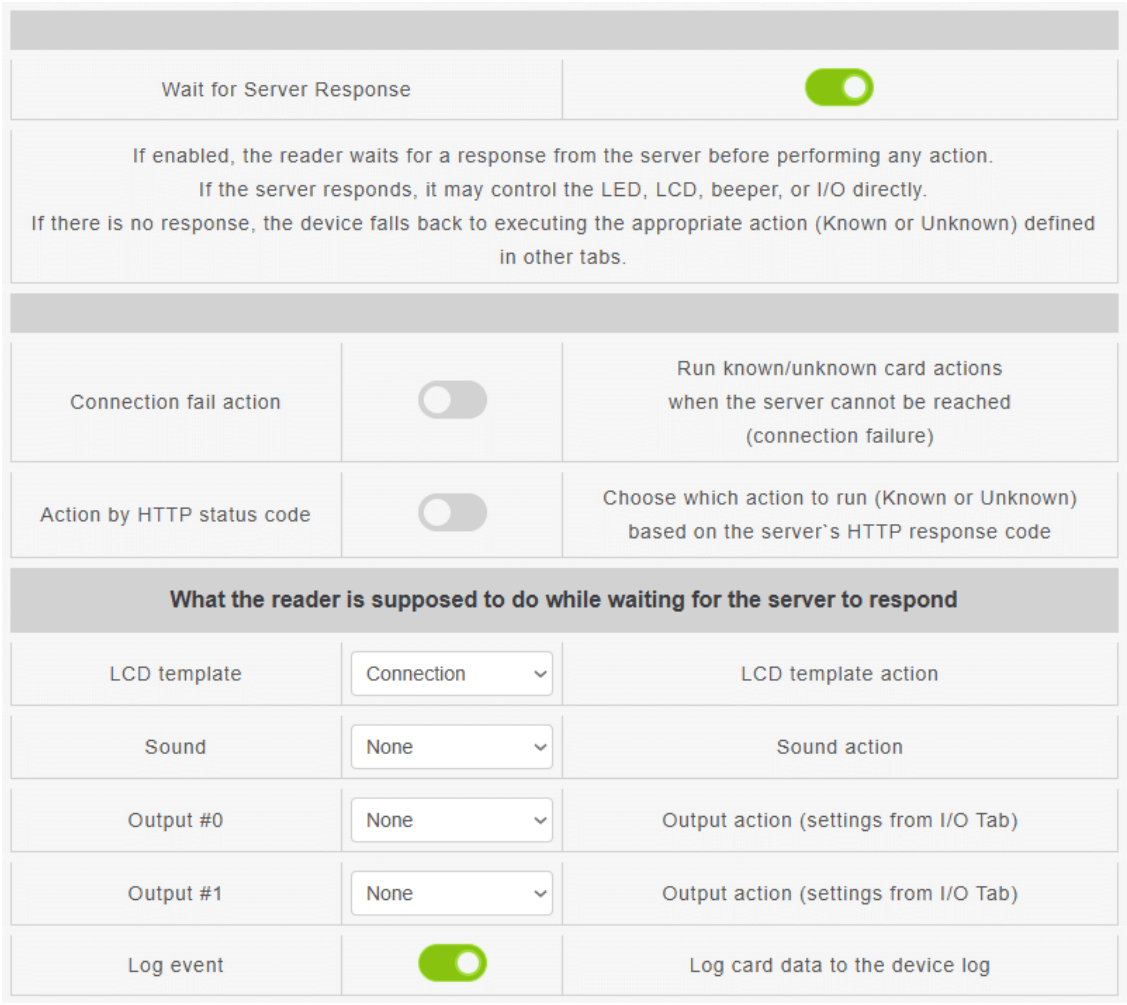
Card Action

The **Card Action** section allows for configuration of reader's reaction for RFID tag application.

On-new

The **Action on server requests** section enables the user to define the reader's behaviour while waiting for a reply from server or while receiving it.





- **HTTP Request** – Enable the **On-new card** service,
  - **HTTP Method** – HTTP communication method: **GET, POST, PUT, DELETE**,
  - **Content type** – Header specifying the type of transferred data (eg. application/json, application/x-www-form-urlencoded),
  - **HTTP request** – HTTP request syntax field.
- **MQTT** – Enable the **MQTT** service:
  - **Publish Topic** – Topic, to which the card read action notification will be sent to. The topic can be static or dynamic,
  - **MQTT payload** – MQTT notification syntax.
- **Wait for server response** – If the option is enabled, the reader waits for server reply and executes returned commands (controlling the LED, LCD, buzzer, I/O). Otherwise, it checks the card database and executes appropriate actions:
  - **Connection fail action** – Set execution of action known/unknown in the event of server communication loss,
  - **Action by HTTP status code** – Set execution of action known/unknown depending on the server HTTP response code,
  - **LCD template** – The template that will be displayed on the LCD screen while waiting for server response,
  - **Sound** – Sound signal:
    - **None** – None,

    - **Accept** – Tag accepted signal,
    - **Reject** – Tag rejected signal,
    - **Tick** – Short sound signal.
- **Output #0** – Activation of relay output No. 0:
    - **None** – Skip,
    - **Off** – Disable relay output,
    - **On** – Enable relay output,
    - **Toggle** – Change of relay state to the opposite,
    - **Period** – Enable the relay output for **Out #0 Time** time period defined in the **I/O Settings** tab,
    - **Long Period** – Enable the relay output for **Out #0 Time Long** time period defined in the **I/O Settings** tab,
    - **User Period** – Enable the relay output for **Output #0 User period**.
- **Output #1** – Activation of relay output No. 1 – Settings similar to output #0,
- **Log event** – Logs the card application events.

Tip

After making changes, save them using the Save button

On-known

This tab allows defining reactions of the reader after a card stored in the device's memory is applied.

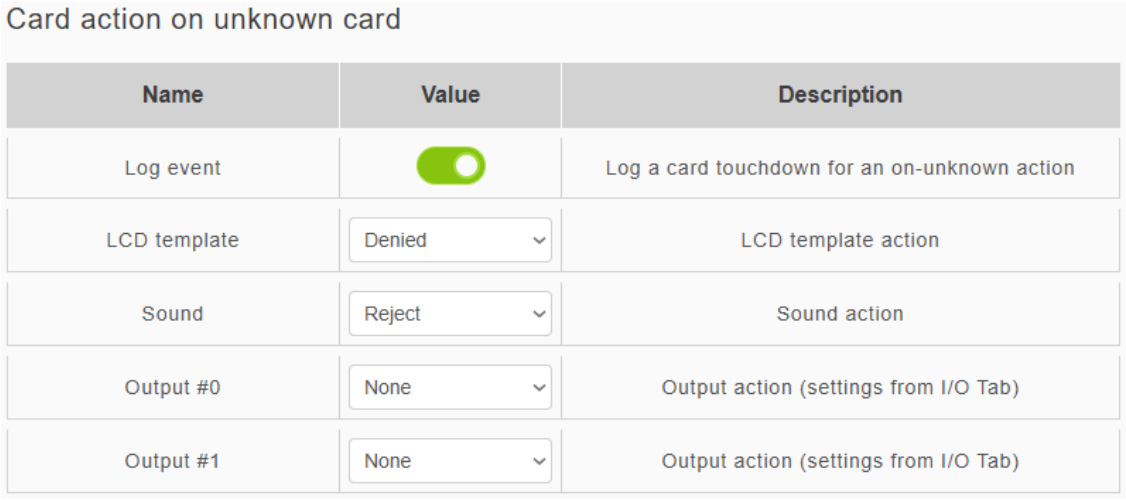
Card action on known card

Name	Value	Description
Log event		Log a card touchdown for an on-known action
LCD template	Granted	LCD template action
Sound	Accept	Sound action
Output #0	Period	Output action (settings from I/O Tab)
Output #1	None	Output action (settings from I/O Tab)

- **Log event** – Enables logging of card application events if action attributed to the card is known and has been defined in the system.
- **LCD template** – The template that will be displayed on the LCD screen in response to the event,
- **Sound** – Sound signal:
  - **None** – None,
  - **Accept** – Tag accepted signal,
  - **Reject** – Tag rejected signal,
  - **Tick** – Short sound signal.
- **Output #0** – Activation of relay output No. 0:
  - **None** – Skip,
  - **Off** – Disable relay output,
  - **On** – Enable relay output,
  - **Toggle** – Change of relay state to the opposite,
  - **Period** – Enable the relay output for **Out #0 Time** time period defined in the **I/O Settings** tab,
  - **Long Period** – Enable the relay output for **Out #0 Time Long** time period defined in the **I/O Settings** tab,
  - **User Period** – Enable the relay output for **Output #0 User period**.
- **Output #1** – Activation of relay output No. 1 – Settings similar to output #0.

On-unknown

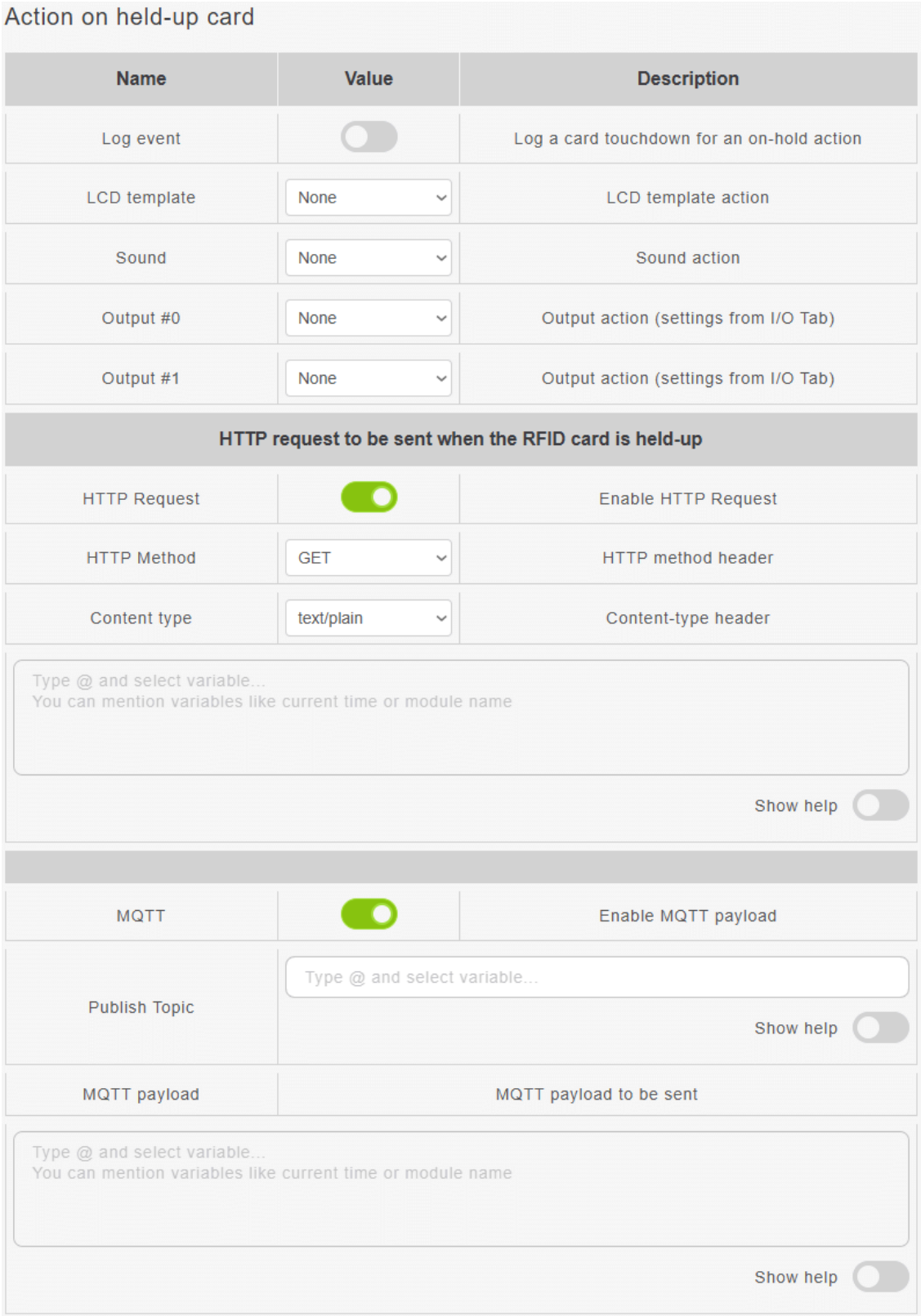
This tab allows defining reactions of the reader after a card that isn't saved or has been deactivated in the device's memory is applied.



- **Log event** – Enables logging if an RFID tag is not recognized.
- **LCD template** – The template that will be displayed on the LCD screen in response to the event,
- **Sound** – Sound signal:
  - **None** – None,
  - **Accept** – Tag accepted signal,
  - **Reject** – Tag rejected signal,
  - **Tick** – Short sound signal.
- **Output #0** – Activation of relay output No. 0:
  - **None** – Skip,
  - **Off** – Disable relay output,
  - **On** – Enable relay output,
  - **Toggle** – Change of relay state to the opposite,
  - **Period** – Enable the relay output for **Out #0 Time** time period defined in the **I/O Settings** tab,
  - **Long Period** – Enable the relay output for **Out #0 Time Long** time period defined in the **I/O Settings** tab,
  - **User Period** – Enable the relay output for **Output #0 User period**.
- **Output #1** – Activation of relay output No. 1 – Settings similar to output #0,

On-hold

This tab allows defining reactions of the reader for holding a card in the read area for time defined in the **Action Hold time** parameter (default 5 seconds). This time can be configured in the **RFID Settings** section.

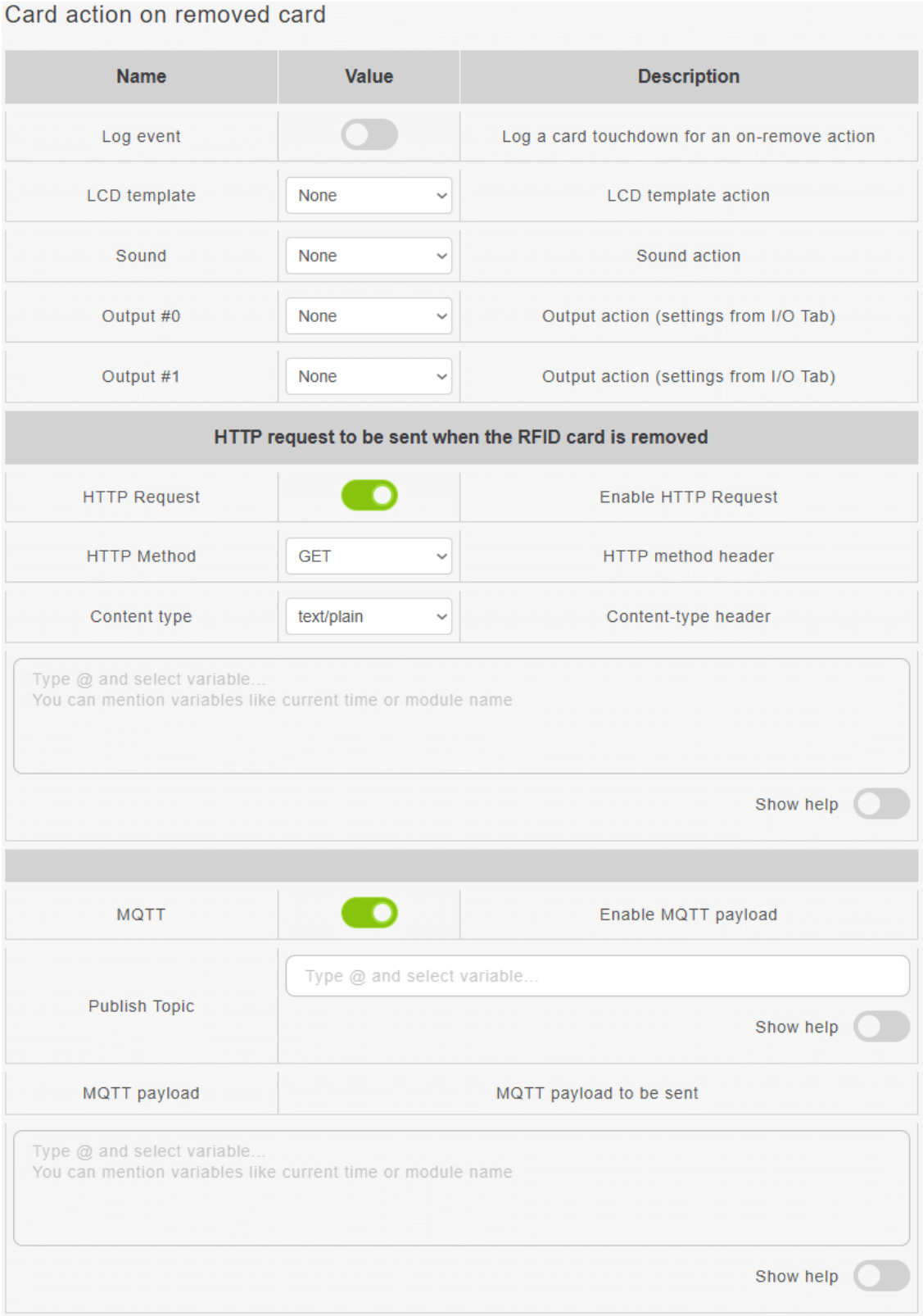


- **Log event** – Enables event logging if an RFID tag is held in range of the reader,
- **LCD template** – The template that will be displayed on the LCD screen if an RFID tag is held in range of the reader for time defined in the **Action Hold time** field,
- **Sound** – Sound signal:

    - **None** – None,
    - **Accept** – Tag accepted signal,
    - **Reject** – Tag rejected signal,
    - **Tick** – Short sound signal.
- **Output #0** – Activation of relay output No. 0:
    - **None** – Skip,
    - **Off** – Disable relay output,
    - **On** – Enable relay output,
    - **Toggle** – Change of relay state to the opposite,
    - **Period** – Enable the relay output for **Out #0 Time** time period defined in the **I/O Settings** tab,
    - **Long Period** – Enable the relay output for **Out #0 Time Long** time period defined in the **I/O Settings** tab,
    - **User Period** – Enable the relay output for **Output #0 User period**.
- **Output #1** – Activation of relay output No. 1 – Settings similar to output #0,
- **HTTP Request** – Enable the HTTP service,
    - **HTTP Method** – HTTP communication method: **GET, POST, PUT, DELETE**,
    - **Content type** – Header specifying the type of transferred data (eg. application/json, application/x-www-form-urlencoded),
    - **HTTP request** – HTTP request syntax field.
- **MQTT** – Enable the **MQTT** service,
    - **Publish Topic** – Topic, to which the card read action notification will be sent to,
    - **MQTT payload** – MQTT notification syntax.

On-remove

This tab allows defining reactions of the reader the moment a card is removed from the read area.



- **Log event** – Enables event logging if an RFID tag is removed from the reader,
- **LCD template** – The template that will be displayed on the screen if an RFID tag is removed,
- **Sound** – Sound signal:
  - **None** – None,

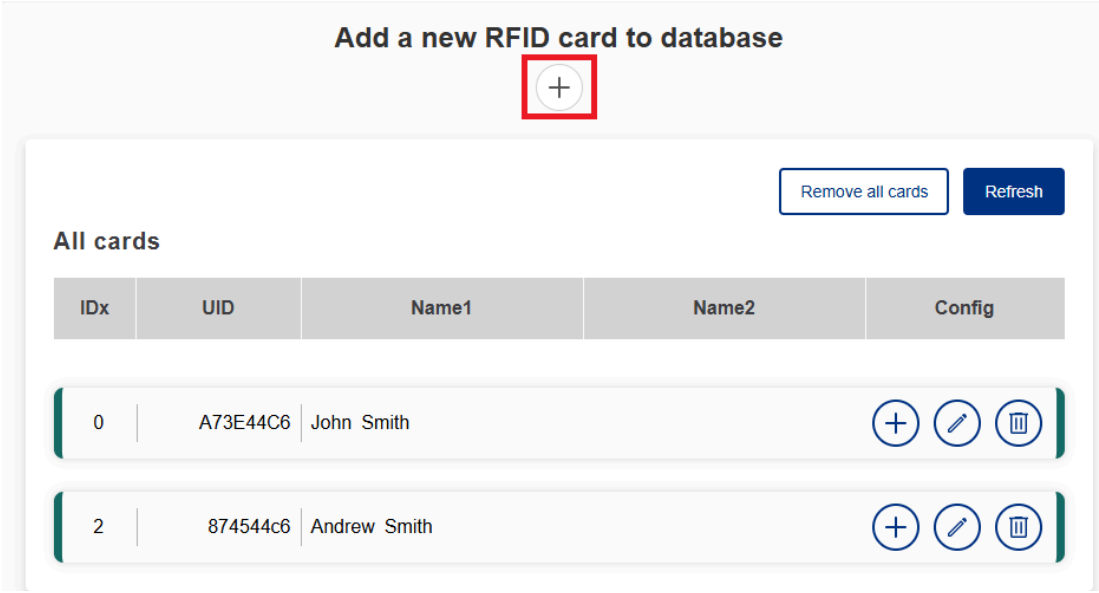
    - **Accept** – Tag accepted signal,
    - **Reject** – Tag rejected signal,
    - **Tick** – Short sound signal.
- **Output #0** – Activation of relay output No. 0:
    - **None** – Skip,
    - **Off** – Disable relay output,
    - **On** – Enable relay output,
    - **Toggle** – Change of relay state to the opposite,
    - **Period** – Enable the relay output for **Out #0 Time** time period defined in the **I/O Settings** tab,
    - **Long Period** – Enable the relay output for **Out #0 Time Long** time period defined in the **I/O Settings** tab,
    - **User Period** – Enable the relay output for **Output #0 User period**.
- **Output #1** – Activation of relay output No. 1 – Settings similar to output #0,
- **HTTP Request**:
    - **HTTP Method** – Method used for communication: **GET, POST, PUT, DELETE**,
    - **Content type** – text/plain, json, xml, form,
    - **HTTP Request** syntax.
- **MQTT** – Enable the **MQTT** service:
    - **Publish Topic** – Topic, to which the notification will be sent,
    - **MQTT Payload** – Notification syntax.

Managing cards and events

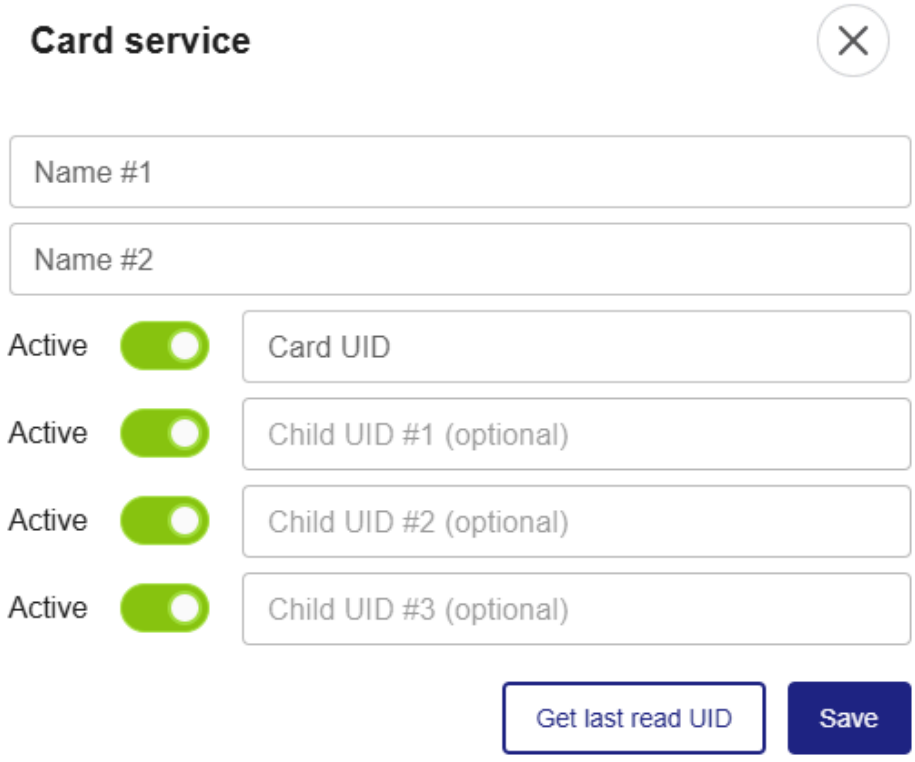
Card database

The RFID reader is equipped with internal memory for managing RFID cards. The user can add cards to the reader both manually and by importing a CSV file. The device's reaction can be different depending on whether a card is present in the reader's memory or not.

The **Card Database** tab in the user interface allows for manual adding, editing and erasing RFID tags from the device's memory.



To add a new user, click **+** and fill out the **Card service** table.

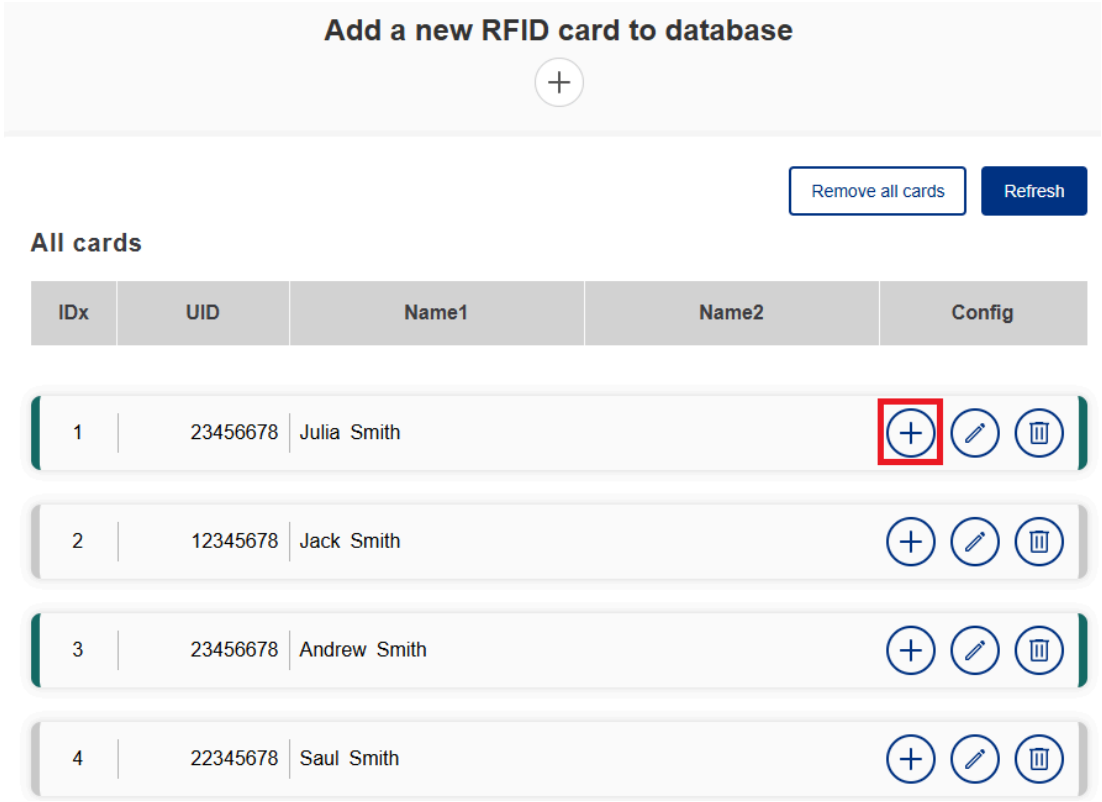


- **Name #1** – User's first name,
- **Name #2** – User's last name,
- **Card UID** – RFID tag UID number,
- **Child UID 1-4** – Child tag UID's,
- **Get last read UID** – Function enters the last UID read by the reader,
- **Save** – Add record to a temporary RFID tag list.

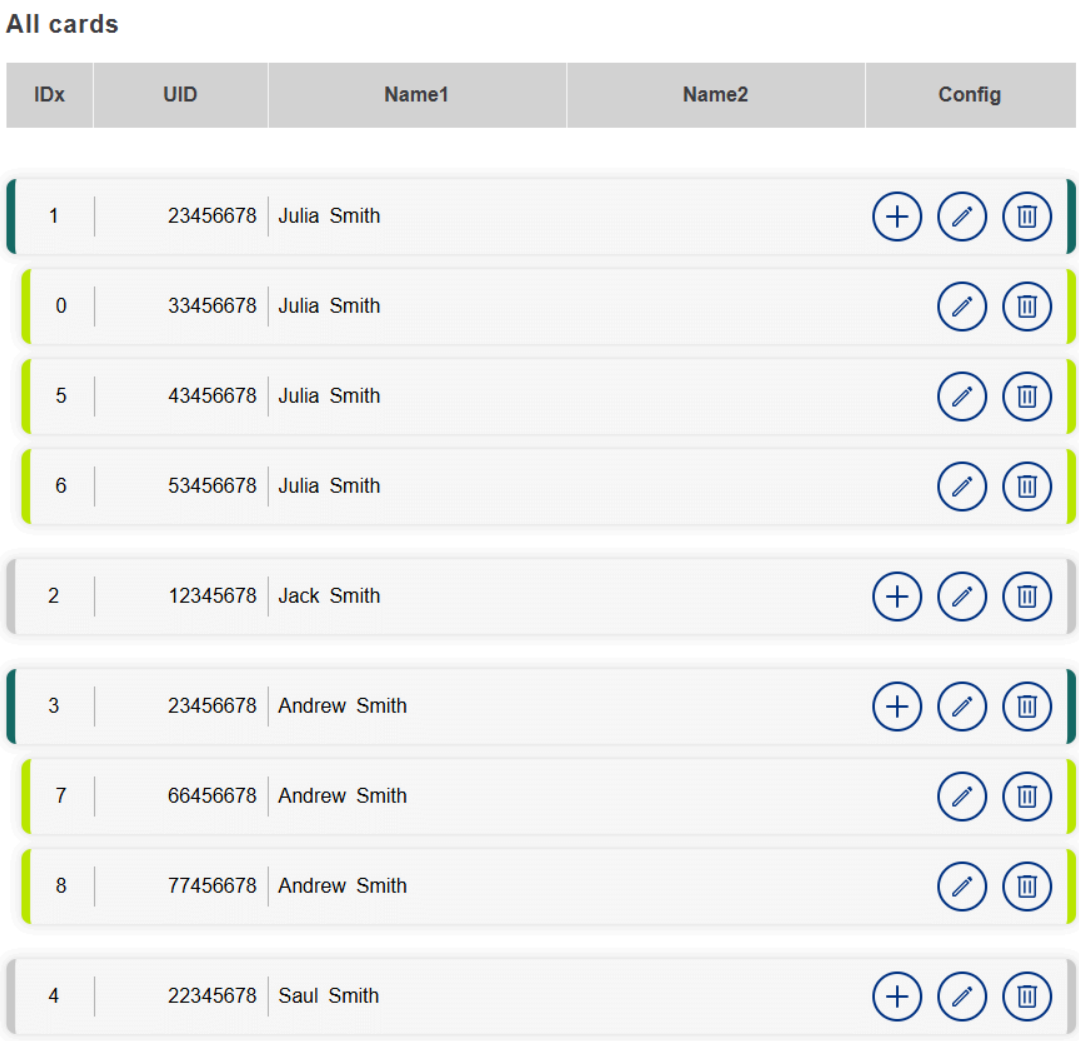
The system allows assignment of multiple RFID cards to a single user. Thanks to that, the user can use multiple cards instead of a single one.

To add additional cards:

1. Go to the user list and find the user you want to assign new cards to,
2. Click the **+** icon by the main card (parent),
3. Enter the new card data and accept.



End result:



Tip

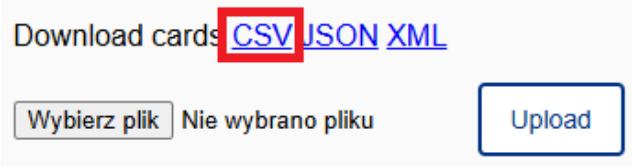
The save button causes a creation of a temporary RFID tags list, which is not immediately saved to the non-volatile memory. If a tag has been added or a record amended, a notice to save data using "Save here!" will appear in the top section of the screen.

Information

The cards are colour-coded according to their status:
- Dark green – Main cards (parent),
- Light green – Sub-cards (child),
- Grey – Deactivated cards.

CSV upload

Located at the bottom of the **Card Database** page is an option to upload the card database in the form of a CSV file.



Warning

Size limit: maximum size of the uploaded file is 200 kB, which equals to roughly 5000 users.

Before uploading the file, make sure that its structure complies with the following example:

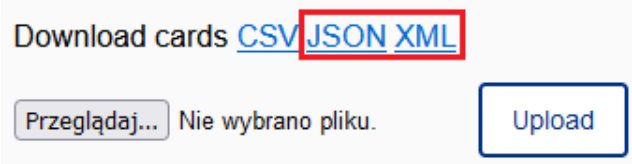
```
uid;name1;name2;idx;parent;flag
123456789;John;Smith;0;-1,1
87654321;Saul;Smith;1;-1,1
11223344;;;2;0;1
```

The first row contains a header (ignored during file upload):

- **uid** – RFID tag ID,
- **name1** – Name1 field from the card database (empty for a Child card),
- **name2** – Name2 field from the card database (empty for a Child card),
- **idx** – Unique user ID,
- **parent** – ID of main card (Parent) or `-1` if the card is the main one (isn't associated with another card as a sub-card),
- **flag** – Value of the **Enable card** field from the card database. `1` for an active card, `0` for an inactive card.

Downloading the database

The internal card database can be downloaded in the form of CSV, JSON or XML files. To gain access to those files, refer to appropriate resources (e.g. `192.168.111.15/card.csv`) or click an icon at the bottom of the page.



CSV file – `card.csv`

```
uid;name1;name2;idx;parent;flag
12345678;John;Smith;0;-1;1
87654321;Saul;Smith;1;-1;1
11223344;;;2;0;1
```

JSON file – `card.json`

```
{
"card":[
    {
        "uid":"12345678",
        "name1":"John",
        "name2":"Smith",
        "idx":"0",
        "parent":"-1",
        "flag":"1"
    },
    {
        "uid":"87654321",
        "name1":"Saul",
        "name2":"Smith",
        "idx":"1",
        "parent":"-1",
        "flag":"1"
    },
    {
        "uid":"11223344",
        "name1":"",
        "name2":"",
        "idx":"2",
        "parent":"0",
        "flag":"1"
    }
]
}
```

XML file – `card.xml`

```
<card>
    <entry>
        <uid>12345678</uid>
        <name1>John</name1>
        <name2>Smith</name2>
        <idx>0</idx>
        <parent>-1</parent>
        <flag>1</flag>
    </entry>
    <entry>
        <uid>87654321</uid>
        <name1>Saul</name1>
        <name2>Smith</name2>
        <idx>1</idx>
        <parent>-1</parent>
        <flag>1</flag>
    </entry>
    <entry>
        <uid>11223344</uid>
        <name1/>
        <name2/>
        <idx>2</idx>
        <parent>0</parent>
        <flag>1</flag>
    </entry>
</card>
```

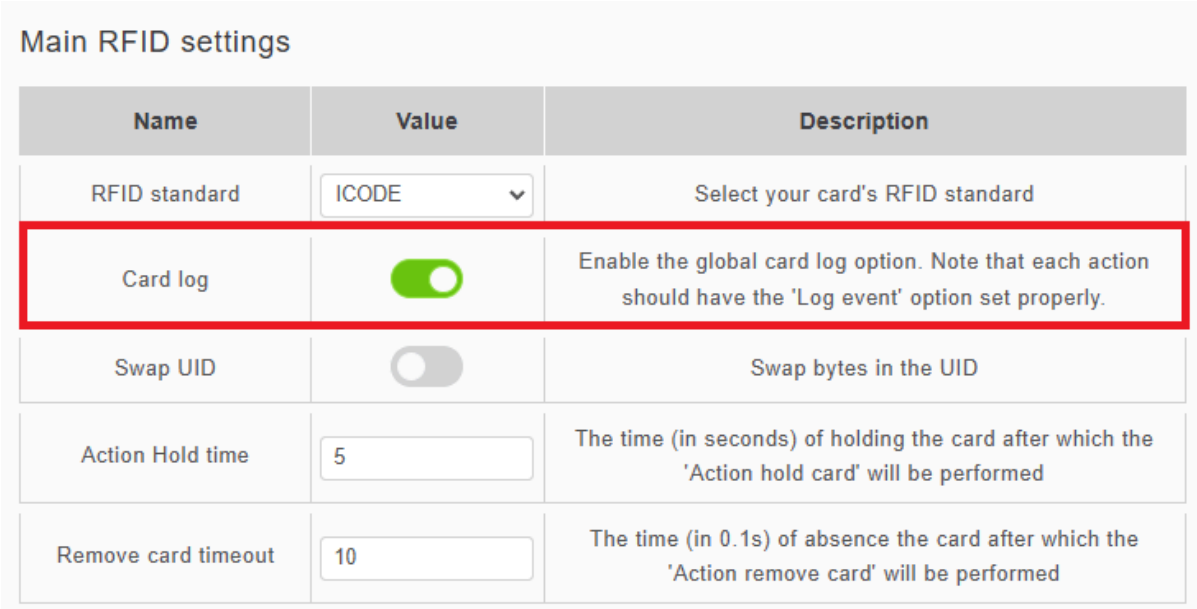
- **uid** – RFID tag ID,
- **name1** – Name1 field from the card database (empty for a Child card),
- **name2** – Name2 field from the card database (empty for a Child card),
- **idx** – Unique user ID,
- **parent** – ID of main card (Parent) or `-1` if the card is the main one (isn't associated with another card as a sub-card),
- **flag** – Value of the **Enable card** field from the card database. `1` for an active card, `0` for an inactive card.

Event logging

The device can register events related to application, holding or removal the card from the reader read area. The event log is available:

- On the website,
- In CSV format,
- In XML format,
- In JSON format.

> **Tip**
>
> **Activation of event logging is possible in the RFID Settings tab, Setup table.**

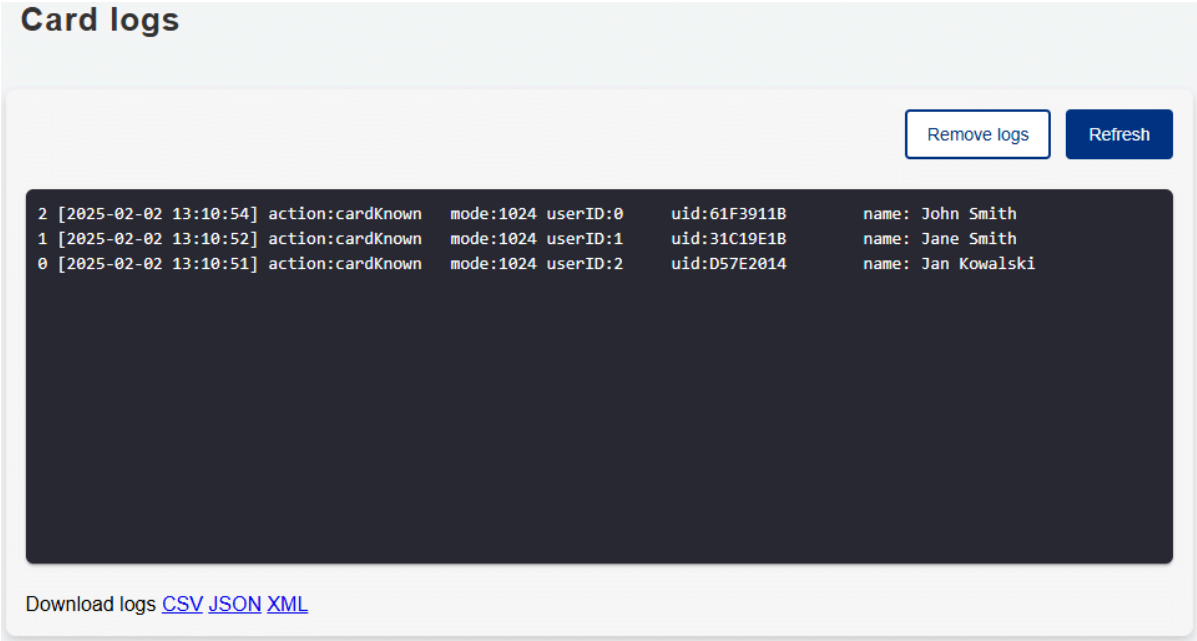


Types of logged events:

- **Action Known Card** – Application of an RFID card stored in the reader's memory,
- **Action Unknown Card** – Application of an RFID card not stored or deactivated in the reader's memory,
- **Action Hold Card** – Holding an RFID card in the read area for minimum **Action Hold Time**,
- **Action Remove Card** – Removal of RFID card outside of the read area,
- **HTTP/MQTT Communication** – Utilizing and control by HTTP or MQTT communication protocols, optionally after enabling **Wait for Server Replay**.

Events preview

The **Card logs** tab allows the user to preview logged events via the device's webpage.



Column No.	Description
1	Sequential number
2	Date and time of the recorded event
3	**Action flag** – meaning dependent on value: **1** – RFID tag read during protocol control (HTTP/...). **2** – action for card stored in the device's memory. **4** – action for unknown/inactive card. **8** – action for hold card function. **16** – action for remove card function
4	**Mode value** – decimal total of events – consult table below
5	**ID number** of the user in card memory. **-1** means that a card is unknown or inactive
6	**UID** of the read **RFID** tag
7	Combined value of **Name** and **Surname** fields from card memory
8	Decimal total of entrance flag values: **ENTER_FLAG_OPT** – Optional 0d1, 0x1 **ENTER_FLAG_IN** – Entrance 0d4, 0x4 **ENTER_FLAG_OUT** – Exit 0d8, 0x8 **ENTER_FLAG_EXT** – Application of card to an external reader 0d16, 0x10 **ENTER_FLAG_KBD** – Record entered with a keypad *(Clocker CL-30 only)* 0d32, 0x20

The Mode value is a sum of the values assigned to the current input / output state:

Input/Output state	Description	binary	dec	hex
LOG_OUT0	Current state of output 0	0100 0000 0000	1024	400
LOG_OUT1	Current state of output 1	1000 0000 0000	2048	800
LOG_IN0	Current state of input 0	01 0000 0000 0000	4096	1000
LOG_IN1	Current state of input 0	10 0000 0000 0000	8192	2000

Information

Logs can be deleted by clicking the Remove logs button.



Log download

The logs can be downloaded in the form of CSV, XML or JSON files. To gain access to those files, refer to appropriate resources (e.g. `192.168.111.15/card.csv`) or click an icon at the bottom of the page.

CSV file – `log.csv`**

```
timestamp;time;date;action;user_id;uid;name1;name2;enter
1782368350;06:19:10;2026-06-25;2;0;12345678;John;Smith;1024
1782368355;06:19:15;2026-06-25;2;1;87654321;Saul;Smith;1024
```

JSON file – `log.json`

```
{
"logs":[
    {
        "action":2,
        "mode":1024,
        "ts":1782368350,
        "time":"06:19:10",
        "date":"2026-06-25",
        "userId":0,
        "uid":"12345678",
        "name1":"John",
        "name2":"Smith"
    },
    {
        "action":2,
        "mode":1024,
        "ts":1782368355,
        "time":"06:19:15",
        "date":"2026-06-25",
        "userId":1,
        "uid":"87654321",
        "name1":"Saul",
        "name2":"Smith"
    }
]
}
```

XML file `log.xml`

```xml
<logs>
    <entry>
        <action>2</action>
        <timestamp>1782368350</timestamp>
        <time>06:19:10</time>
        <date>2026-06-25</date>
        <userId>0</userId>
        <uid>12345678</uid>
        <name1>John</name1>
        <name2>Smith</name2>
        <mode>1024</mode>
    </entry>
    <entry>
        <action>2</action>
        <timestamp>1782368355</timestamp>
        <time>06:19:15</time>
        <date>2026-06-25</date>
        <userId>1</userId>
        <uid>87654321</uid>
        <name1>Saul</name1>
        <name2>Smith</name2>
        <mode>1024</mode>
    </entry>
</logs>
```

- **timestamp/ts** – Timestamp zdarzenia,
- **time** – Czas zdarzenia,
- **date** – Data zdarzenia
- **action**:
  - **1** – Odbicie tagu RFID przy sterowaniu przez protokół,
  - **2** – Akcja dla karty zapisanej w pamięci urządzenia,
  - **4** – Akcja dla karty nieznanej/nieaktywnej,
  - **8** – Akcja dla funkcji **Hold card**,
  - **16** – Akcja dla funkcji **Remove card**.
- **user_id/userId** – ID użytkownika z pamięci kart. Wartość `-1` oznacza kartę nieznaną lub nieaktywną,
- **uid** – UID odczytanego tagu,
- **name1, name2** – Pola **Name1** lub **Name2** z bazy kart,
- **enter/mode** – Zsumowana wartość dziesiętna flag zdarzeń.

Communication protocols

The device supports various communication protocols that can be adapted to individual needs. The configuration is available in the **Services** tab.

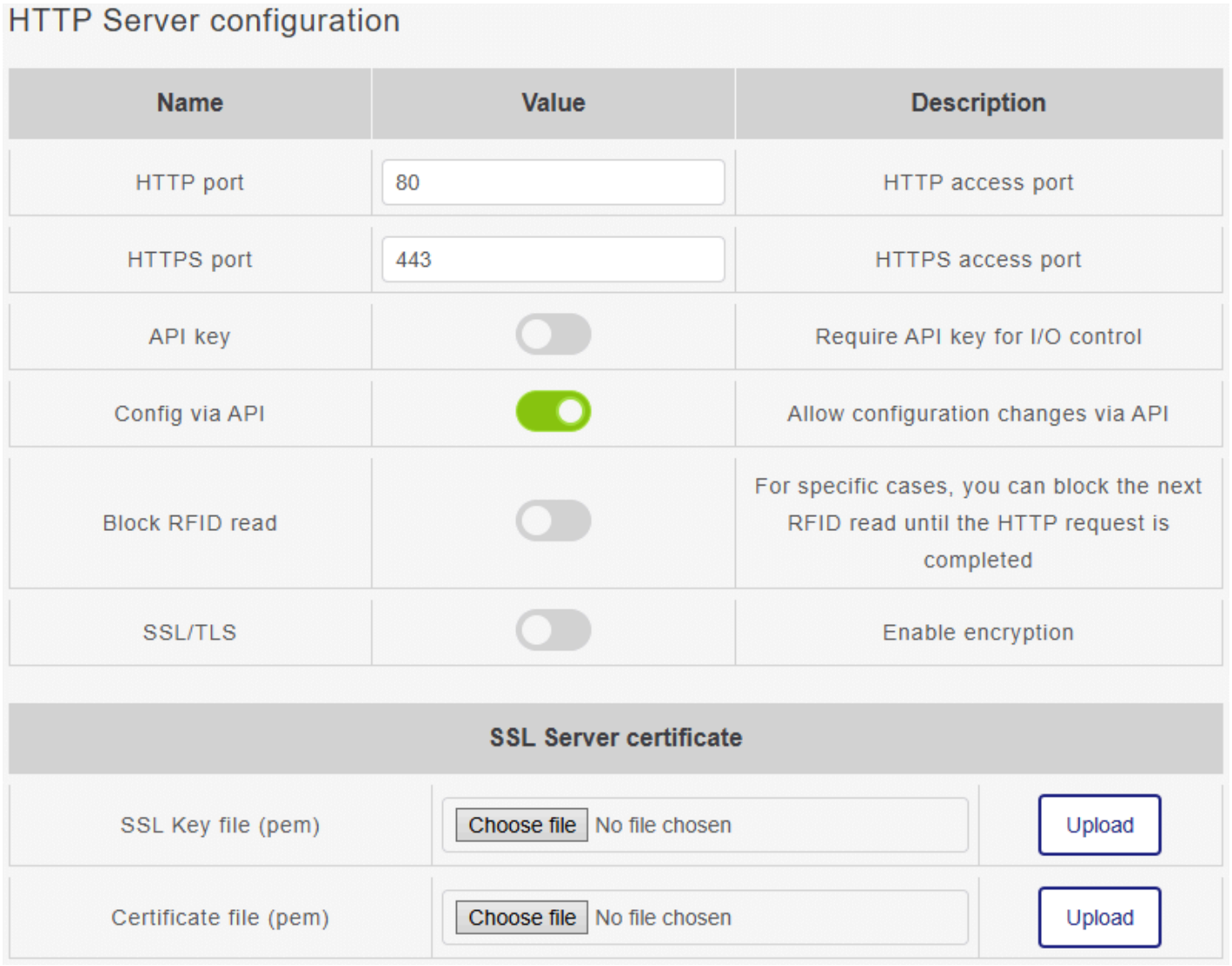
Web – HTTP/HTTPS Server

The reader can be configured as an HTTP or HTTPS server. In this mode, an external client connects with the reader and manages it with HTTP/HTTPS requests (GET method).
The client regularly downloads the `status.xml` or `status.json` file, which contains information about device condition such as:

- **Card UID**,
- **Reader's MAC address**,
- **Input and output states**.

After reading and decoding this data, the client can send commands to the reader, that will recall appropriate reactions, such as:

- **Sound signalling** (eg. card accepted/rejected),
- **LED indicators flashing**,
- **Door lock release**,
- **Displaying text on the screen**.



- **HTTP port** – HTTP server and device webpage access port,
- **HTTPS port** – HTTP server and device webpage access port (SSL/TLS encryption required),
- **API key** – Enable/disable API key requirement for control,
- **Config via API** – Allow for configuration changes via REST API,
- **SSL/TLS** – Enable encrypted HTTPS connection support,
- **SSL Key file (pem)** – SSL key file location (pem format),
- **Certificate file (pem)** – CSR (Certificate Signing Request) file location (pem format).

HTTP GET – Status

To get the current module status, recall the resource under the address: `<IP_address>/data/status.xml` or `<IP_address>/data/status.json`.

In response, the resource will be displayed in the form of an XML or JSON file, containing basic information about the reader:

Example response in XML format

```xml
<status>
    <model>Proxer SE</model>
    <name/>
    <mac>00:00:00:00:00:00</mac>
    <id/>
    <newId>0</newId>
    <rfid_enable>1</rfid_enable>
    <cnt>0</cnt>
    <last_read_timestamp>0</last_read_timestamp>
    <n_logs>0</n_logs>
    <in0>0</in0>
    <in1>0</in1>
    <cnt0>0</cnt0>
    <cnt1>0</cnt1>
    <out0>0</out0>
    <out1>0</out1>
    <custom>0</custom>
    <fw>0.13.1</fw>
    <userdb_dirty>0</userdb_dirty>
    <time>10:53:30</time>
    <date>24-02-2026</date>
</status>
```

Example response in JSON format


```json
{
"model": "Proxer SE",
"name": "",
"mac": "00:00:00:00:00:00",
"id": "",
"newId": 0,
"rfid_enable": 1,
"cnt": 0,
"last_read_timestamp": 0,
"n_logs": 0,
"in0": 0,
"in1": 0,
"cnt0": 0,
"cnt1": 0,
"out0": 0,
"out1": 0,
"custom": 0,
"fw": "0.13.1",
"userdb_dirty": 0,
"time": "10:53:30",
"date": "24-02-2026"
}
```


Tag	Description
`model`	Device model
`name`	User-assigned module name

`mac`	Device MAC address
`id`	ID of the last RFID tag read
`newId`	Flag informing about new tag read - read flag
`rfid_enable`	RFID antenna status
`cnt`	Amount of read RFID tags since last device restart
`last_read_timestamp`	Timestamp of the last card read event
`n_logs`	Current amount of logs in the device
`in0`	Current state of input No. 0
`in1`	Current state of input No. 1
`cnt0`	Current counter value for input No. 0
`cnt1`	Current counter value for input No. 1
`out0`	Current state of relay output No. 0
`out1`	Current state of relay output No. 1
`custom`	Special device version flag
`fw`	Device firmware version
`userdb_dirty`	Flag informing about the fact that user records are present in RAM, but not in non-volatile memory
`time`	Current device time
`date`	Current device date

HTTP GET – Control

Tips

1. All HTTP GET commands are mapped to XML/JSON tags for HTTP Client service.
2. Consecutive commands in HTTP GET are to be separated by "&".

All IO commands start with `/io?`

Command	Value	Description
out_run	INTEGER	Activate output in default mode
out_on	INTEGER	Output activation. eg. `"out_on":1` activates output No. 1
out_off	INTEGER	Output deactivation
txt0	STRING	Internal variable for controlling the display
txt1	STRING	As above
txt2	STRING	As above
txt3	STRING	As above
template_index	INTEGER	Template index to be displayed
template_name	STRING	Template name to be displayed
buzz	INTEGER	Sound emission: 1 – accept, 2 – reject, 3 - short signal example: `"buzz":2` - buzzer will emit the card rejection sound signal
userdb_ver	STRING	Returns the card database stored in RAM (if the base is sent to the reader via HTTP Client). If the card database is updated dynamically or dependent on specified conditions, this command allows to monitor its version and check if changes have been made
userdb_store	INTEGER	Transcription of card data from **RAM** to **FLASH** (**1**). This operation should be performed only after all user modifications have been made, and as rarely as possible. It should be noted that the amount of guaranteed memory write cycles is 10 000
cardlog_remove_n	INTEGER	Delete last x card logs
cardlog_remove_all	BOOLEAN	Delete all card logs
enter_state	**FLAG_OPT=1** **FLAG_IN=4** **FLAG_OUT=8** **FLAG_EXT=10** **FLAG_KBD=20**	

	FLAG_QR=40 ***(HEX)***	**Setting of entry mode. Returns the value in decimal format.** **FLAG_OPT (0d1, 0x01):** Optional condition (e.g. private/work) **FLAG_IN (0d4, 0x04):** Sets the mode to entrance – records the entrance time. **FLAG_OUT (0d8, 0x08):** Sets the exit mode – records the exit time. **FLAG_EXT (0d16, 0x10):** Card application to an external reader. **FLAG_KBD (0d32, 0x20):** Keypad record *(Clocker CL-30 only)* **FLAG_QR (0d64, 0x40):** Built-in QR code reader record *(option)*. **Example:** The setting for **optional entrance** has a value of **5**, which is the result of combining **FLAG_OPT (0d1)** with **FLAG_IN (0d4)**
rfid_en	BOOLEAN	Enable (**1**) or disable (**0**) card reading

HTTP GET – RFID card database

> **Tip**
>
> **After a power cycle, the reader transcribes the card data from flash memory to RAM.**

> **Warning**
>
> **After all edits have been made, it's needed to write all data to the flash memory. The flash memory write procedure should be performed as rarely as possible.**

All DB commands start with `/db?`

Command	Value	Description
add	`uid;name1;name2;idx;parent;flag`	Add card entry in CSV format. The data must be URL encoded. - If `idx==-1`, an empty location will be found. - If `idx≠-1`, an existing location marked by `idx` will be overwritten. - `parent` marks an existing entry, making an affiliation; value should be set to `-1` if not used
store	1	Write card data from **RAM** to **flash**
remove_all	1	Delete all cards from **RAM**
remove	`idx`	Delete selected card from **RAM**
reload	1	Initialize **RAM** with cards from **flash**
get_by_uid	`uid`	Returns the index with selected card UID. If a card does not extst, returns `-1`

> **Example**
>
> **Add a card `John Smith` to the record 10, UID `1234567890`, no Parent card: `http://<IP_address>/db?add=1234567890;John;Smith;10;-1;1`**

HTTP Client

In the **HTTP Client** mode, the module operates similarly to a web browser – it initiates a connection with a server, sending requests and awaiting a reply.

Main features:

- **Automatic data transfer:** After an RFID tag is read, the module automatically connects to the server and transfers information about the tag,
- **Receiving commands from the server:** The server can send module control instructions such as: generate sounds or display text on the LCD screen,
- **Easy integration:** The mode allows for easy integration with various server technologies such as PHP, Node.js or MySQL.

Benefits:

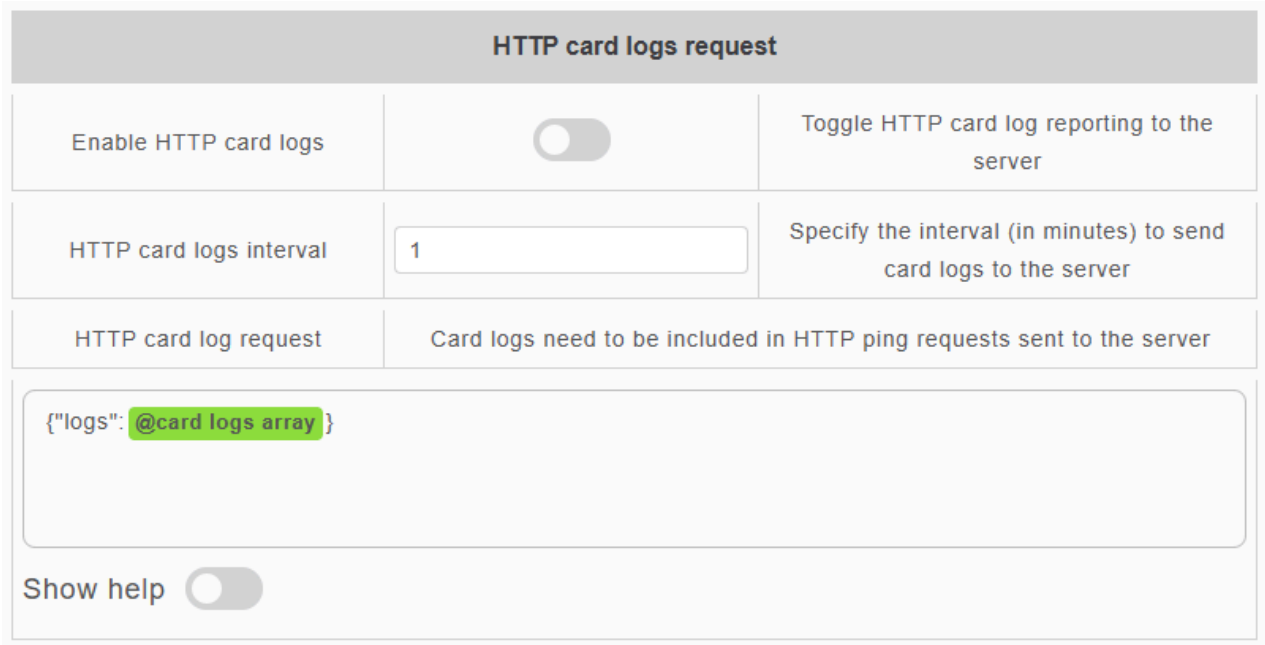
- **Instantaneous communication:** The module automatically initiates a connection after a card is read, allowing for quick data transfer to the server,
- **Flexible control:** The server can control different module elements in response to the commands sent.

HTTP Client configuration

HTTP Client configuration

Name	Value	Description
HTTP Client		Enable HTTP Client
Server		Remote server address: IP or DNS-registered name
HTTP port	0	HTTP access port
Connection timeout	5000	Timeout for the server response [ms] min: 1000 ms, max: 10000 ms
HTTP Method	POST	HTTP request default method
Content type	json	Content-type header
Resource	/	HTTP resource i.e. /resource.php
API key		Require API key for I/O control
Authentication to Server		
Authentication type	Bearer Token	Select the authentication method for server communication
API Key		Bearer Token for server authentication
HTTP ping request		
HTTP ping request interval	0	Ping time in secs. 0-disable
HTTP ping request	HTTP ping requests to test the availability of web servers and APIs. These can be used in health testing or monitoring systems.	

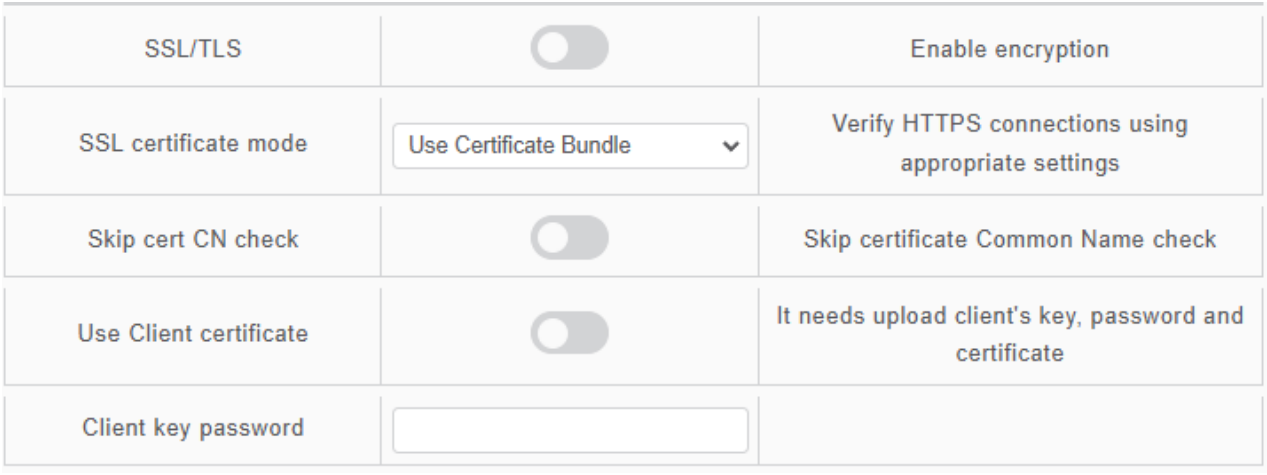
- **HTTP Client** – Enable the **HTTP Client** service,
- **Server** – IP address of a server the module initiates a connection with,
- **HTTP port** – Port, on which the server listens (default **80** for HTTP or **443** for HTTPS),
- **Connection timeout** – (in ms),
- **HTTP Method** – HTTP method used for communicating: **GET, POST, PUT, DELETE**,
- **Content type** – Header determining the type of transferred data (eg. text/plain, json, xml, form),
- **Resource** – Access path to a server resource the module communicates with,
- **API key** – Enable/disable API key requirement for control,
- **User** – User name (if using server authentication),
- **Password** – User password (if using server authentication),
- **HTTP ping request interval** – Server query frequency (0 - off),
- **HTTP ping request** – Server query request syntax.



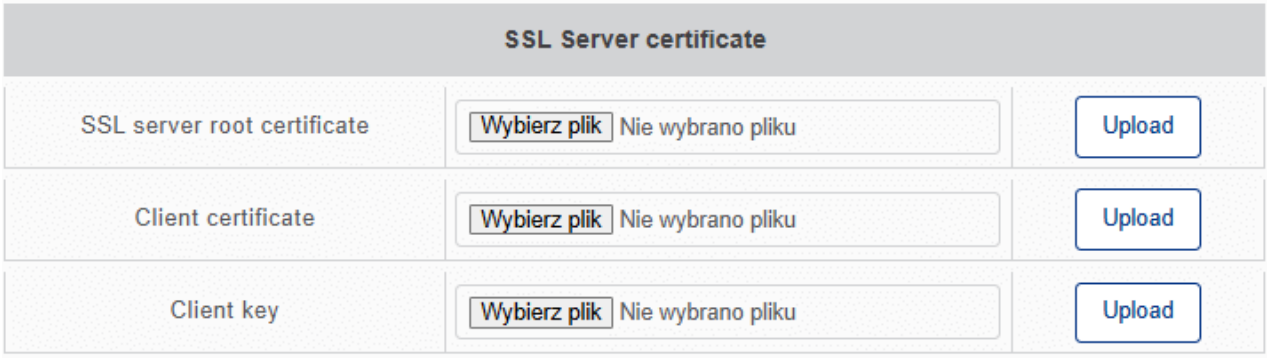
- **Enable HTTP card logs** – Sends card logs to the server. An empty log won't be transfered,
- **HTTP card logs interval** – Log sending interval,
- **HTTP card log request** – Request syntax. The card logs variable must be contained in the request sent to the server.

Information

All certificate files required for establishing an encrypted connection can be transferred in the SSL Client Certificate section.



- **SSL/TLS** – Enable encryption using SSL/TLS protocols,
- **SSL certificate mode**:
  - **Use Certificate Bundle** – Use the built-in certificate bundle,
  - **Use Uploaded Certificate**,
  - **Disable SSL verification**.
- **Skip cert CN check** – Skip checking of CN (Common Name) field compatibility server certificate,
- **Use Client certificate** – Enable use of client certificate to authenticate on the server side,
- **Client key password** – Password securing the client key.



HTTP Client – poll request

The reader can periodically send poll requests to the server, in order to check the status or keep the connection open. The content of such request is defined in the **HTTP Ping Request** field on the **Services / HTTP Client** page. The request can contain both static text and predefined variables, which allows for dynamic adaptation to its contents.

Example: Sending a "poll" request to the resource every 6 seconds:

- **GET request** using predefined variables

`ip=%eip%&name=%mod_name%&in1=%in1%`

Data received on the server side:

```
GET /resource?ip=192.168.0.176&name=T&in1=1 HTTP/1.1
User-Agent: Proxer SE
Host: 192.168.0.90
Content-Length: 0
```

The server can send data back in the form of a JSON (XML) file:

```
HTTP/1.1 200 OK
Content-Length: 38
Content-Type: application/json
{"lcd_text1":"Place an RFID tag"}
```

- **POST request** using predefined variables:

```
ip=%eip%
name=%mod_name%
in1=%in1%
```

Data received on the server side:

```
OST / HTTP/1.1
User-Agent: Proxer SE
Host: 192.168.0.90
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
ip=192.168.0.176
name=T
in1=0
```

The server can send data back in the form of a JSON (XML) file:

```
HTTP/1.1 200 OK
Content-Length: 14
Content-Type: application/xml
<buzz>3</buzz>
```

HTTP Client – Card Action event request

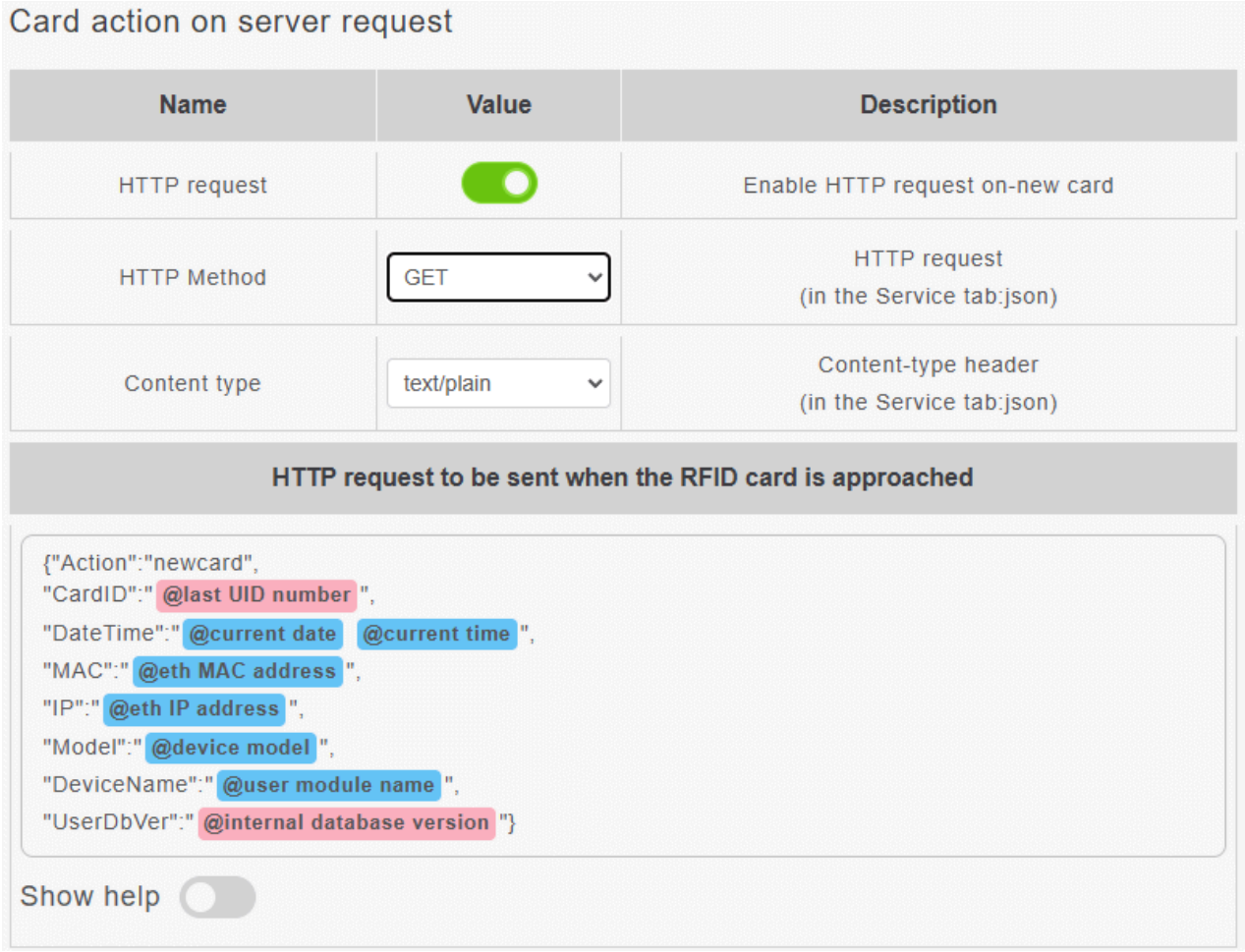
Application of an RFID tag to the reader can trigger sending of a HTTP request to the configured server.

The **Card Action** page allows configuration of three HTTP request send triggers:

- **On-new** – Sends the request after each application of an RFID card to the reader,
- **On-hold** – Sends the request if the card is located in the read area for at least 5 seconds (**hold time** is set to 5 seconds by default),
- **On-remove** – Sends the request if the RFID tag is removed from the read area.

> **Information**
>
> **Each scenario shown above operates independently, which allows the user to choose what events are to trigger HTTP requests. Additionally, the user can individually configure the request contents, its type and a method for each trigger.**

The user configures the request content, its type and a method for each trigger separately.



HTTP request configuration parameters:

- **HTTP Request** – Enable HTTP Request service,
- **HTTP method:** – HTTP method used for communicating: GET, POST, PUT, DELETE,
- **Content type** – Type of header: (eg. text/plain, json, xml, form),
- **HTTP request** – HTTP request syntax. The user defines the full syntax of the request, adjusting it to the server needs (eg. dynamic data, predefined variables).

Example 1

For data from the screenshot and **On new card action** trigger:

- **HTTP Client request**

`mac=%emac%&ip=%eip%&id=%uid%`

After a card is applied, data on the server side will look like this:

```
GET /?mac=00:00:00:00:00:00&ip=192.168.0.176&id=12005CB99F HTTP/1.1
User-Agent: Proxer SE
Host: 192.168.0.90
Content-Type: text/plain
Content-Length: 0
```

The server can return data in the form of a JSON (XML) file:

```
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json
{
"lcd_text1":" Welcome!",
"buzz":"3"
}
```

HTTP Client – Control

> **Tip**
>
> **1. Controlling the module in HTTP Client mode is possible after enabling HTTP Client in the Services tab.**
> **2. All HTTP GET commands are mapped to XML/JSON tags for HTTP Client service**

Command	Value	Description
out_run	INTEGER	Activate output in default mode
out_on	INTEGER	Output activation. eg. `"out_on":1` activates output No. 1
out_off	INTEGER	Output deactivation
txt0	STRING	Internal variable for controlling the display
txt1	STRING	As above
txt2	STRING	As above
txt3	STRING	As above
template_index	INTEGER	Template index to be displayed
template_name	STRING	Template name to be displayed
buzz	INTEGER	Sound emission: 1 – accept, 2 – reject, 3 - short signal example: `"buzz":2` - buzzer will emit the card rejection sound signal
addUser	[{ uid: *STRING*, name1: *STRING* name2: *STRING* idx: *INTEGER* flag: *INTEGER* parent: *INTEGER* }, ...]	Add a JSON format card record to RAM. - If `idx=-1`, the next free location will be found. - If `idx≠1`, an existing location will be overwritten. `flag=` indicates that a card is active. `parent` marks an existing entry, making an affiliation; value should be set to `-1` if not used (default). The `idx`, `flag`, and `parent` variables are not required and should only be sent after their parameters have been changed

renewUser	[{ uid: *STRING*, name1: *STRING* name2: *STRING* idx: *INTEGER* flag: *INTEGER* parent: *INTEGER* }, ...]	Overwrite the entire JSON format card database in RAM. - If `idx=-1`, the first free location will be used. - If `idx≠1`, a specified location will be overwritten. `flag=` indicates that a card is active. `parent` marks an existing entry, making an affiliation; value should be set to `-1` if not used (default). The `idx`, `flag`, and `parent` variables are not required and should only be sent after their parameters have been changed
userdb_ver	STRING	Returns the card database stored in RAM (if the base is sent to the reader via HTTP Client). If the card database is updated dynamically or dependent on specified conditions, this command allows to monitor its version and check if changes have been made
userdb_store	INTEGER	Transcription of card data from **RAM** to **FLASH** (**1**). This operation should be performed only after all user modifications have been made, and as rarely as possible. It should be noted that the amount of guaranteed memory write cycles is 10 000
cardlog_remove_n	INTEGER	Delete last x card logs
cardlog_remove_all	BOOLEAN	Delete all card logs
enter_state	**FLAG_OPT=1** **FLAG_IN=4** **FLAG_OUT=8** **FLAG_EXT=10** **FLAG_KBD=20** **FLAG_QR=40** ***(HEX)***	**Setting of entry mode. Returns the value in decimal format.** **FLAG_OPT (0d1, 0x01):** Optional condition (e.g. private/work) **FLAG_IN (0d4, 0x04):** Sets the mode to entrance – records the entrance time. **FLAG_OUT (0d8, 0x08):** Sets the exit mode – records the exit time. **FLAG_EXT (0d16, 0x10):** Card application to an external reader. **FLAG_KBD (0d32, 0x20):** Keypad record *(Clocker CL-30 only)* **FLAG_QR (0d64, 0x40):** Built-in QR code reader record *(option)*. **Example:** The setting for **optional entrance** has a value of **5**, which is the result of combining **FLAG_OPT (0d1)** with **FLAG_IN (0d4)**
rfid_en	BOOLEAN	Enable (**1**) or disable (**0**) card reading

Example response to overwriting the user database

```
{
userdb_ver: 'test',
renewUser: [
    {
    uid: '12345678',
    name1: 'John',
    name2: 'Smith',
    idx: -1,
    flag: 1,
    parent: -1
    },
    {
    uid: 'ABCDEF01',
    name1: 'Jan',
    name2: 'Kowalski',
    idx: -1,
    flag: 1,
    parent: -1
    }
]
}
```

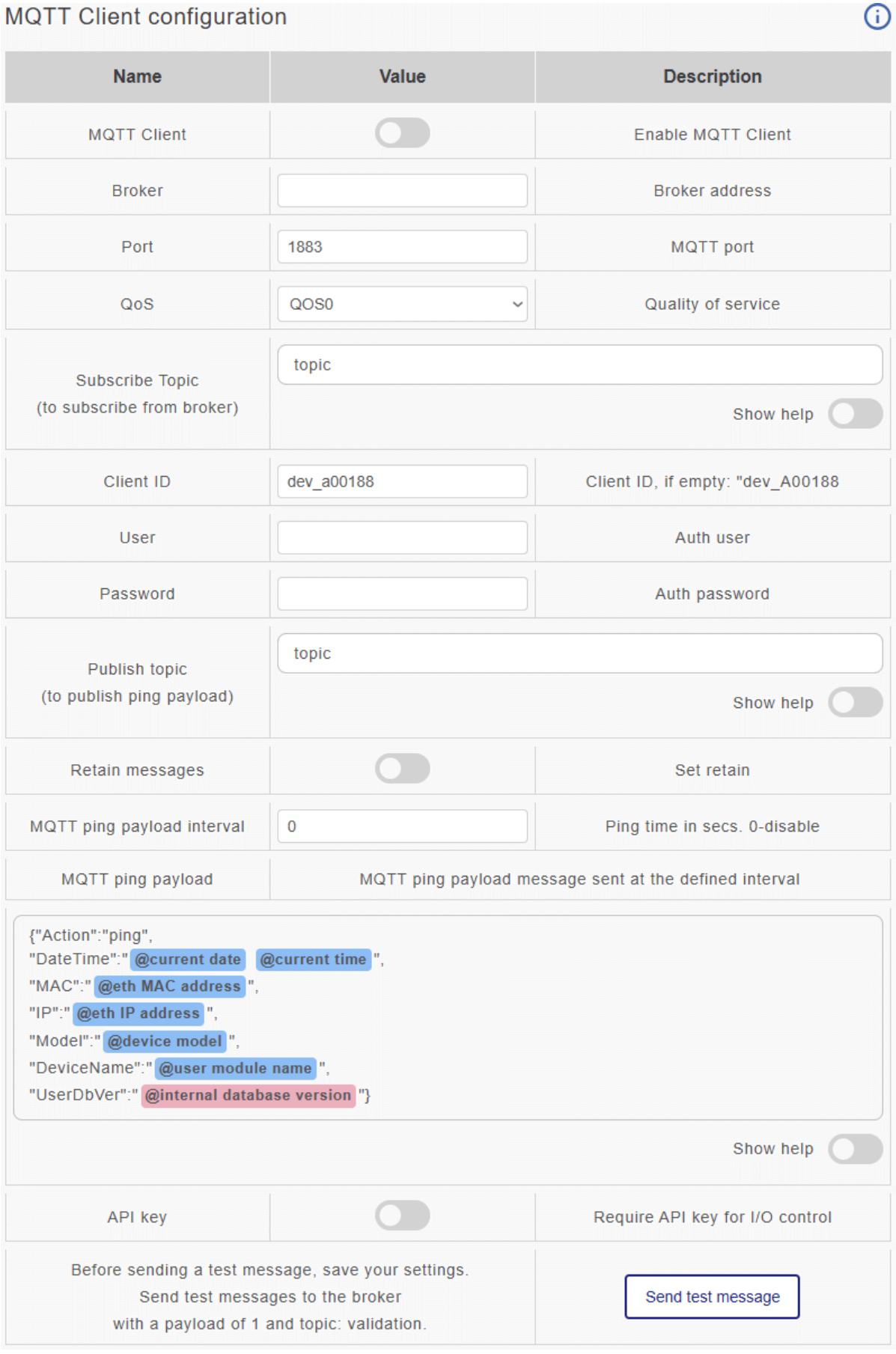
The above response will overwrite the cards in RAM. To save the data to non-volatile memory, use the `userdb_store:1` tag.

Example of php server support

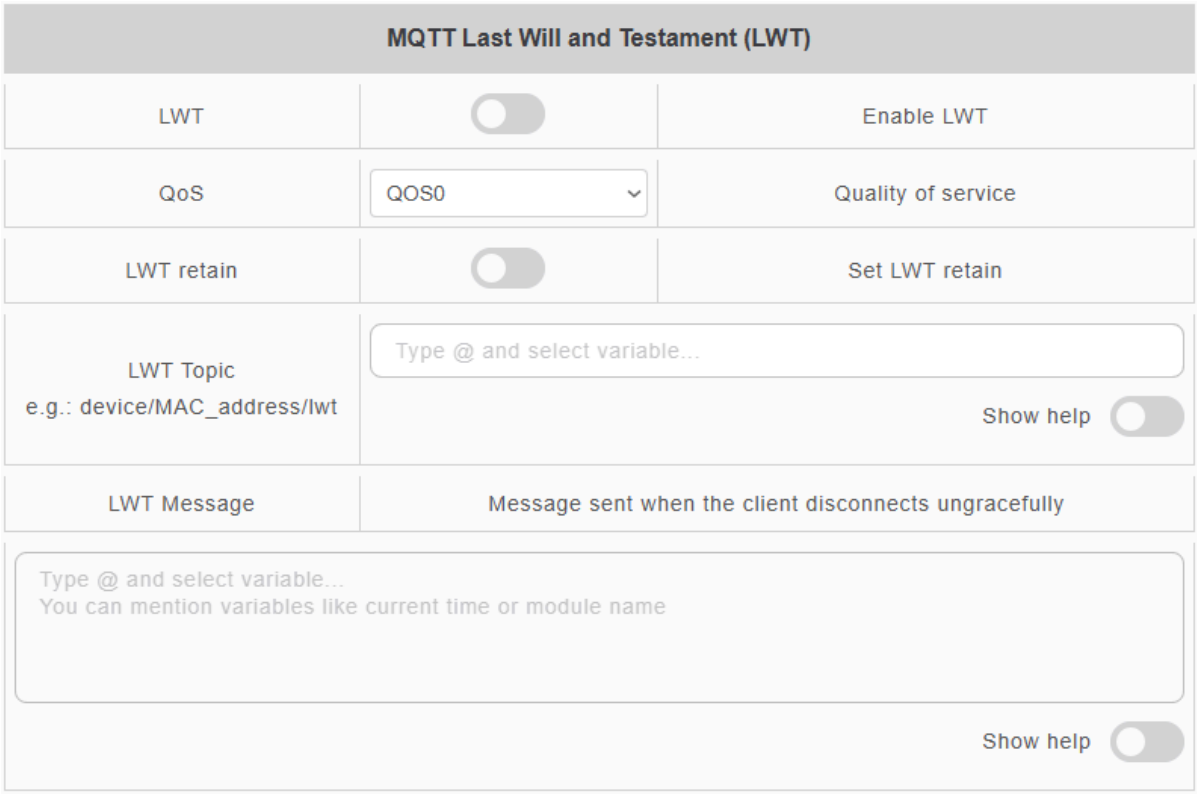
```php
<?php
if( $_GET["id"] ) // receive id and MAC - $_GET["mac"]
{
$who=$_GET["id"];
// check id in DB and do some action
echo "<root>";
echo "<buzz>1</buzz>"; // sound signal
echo "<text>Card ID: $who</text>" // message ID on LCD
echo "<open1>1</open1>"; // door open
echo "</root>";
}
else // no id – poll request
{
echo "<text> Place an RFID tag</text>"; //standby text
}
```

MQTT Client

The device supports the MQTT protocol which allows for data exchange in publication/subscription architecture. The settings can be configured in the **Services** tab, under **MQTT client**.



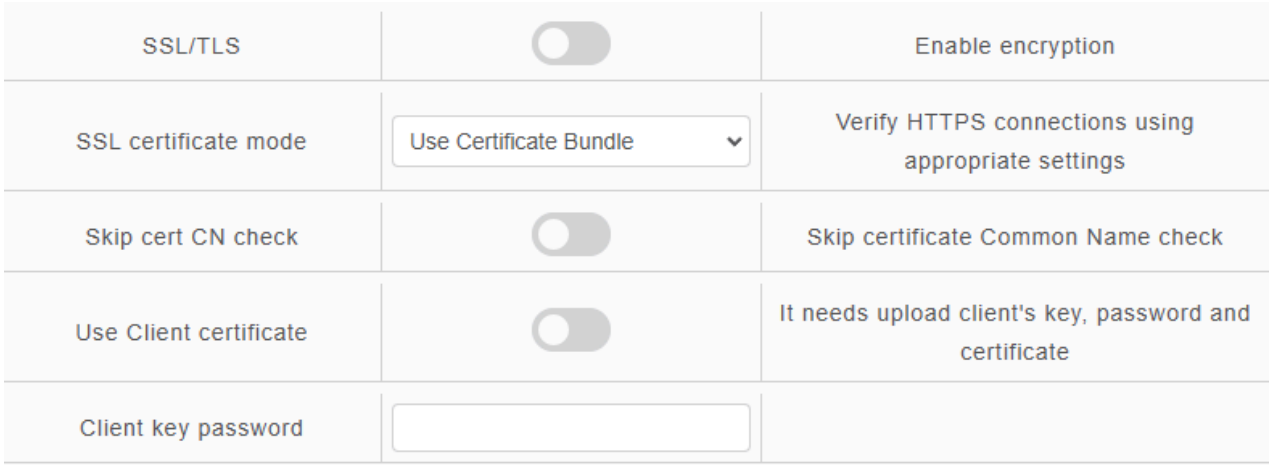
- **MQTT Client** – Enable MQTT service,
- **Broker** – IP address or domain of the MQTT broker,
- **Port** – Port on which port the MQTT server listens (default 1883),
- **QoS** – Quality of Service, supported by the device:
  - **QOS0** – At most once,
  - **QOS1** – At least once,
  - **QOS2** – Exactly once.
- **Subscribe Topic** – Topic, from which the device receives data,
- **Client ID** – MQTT client ID,
- **User** – User name (if authentication is needed),
- **Password** – User password (if authentication is needed),
- **Publish Topic** – Topic to which the device sends data,
- **Retain messages** – If enabled, the message will be saved on the broker an will be available for new topic subscribers,
- **MQTT ping payload interval** – Interval between consecutive ping requests,
- **MQTT ping payload** – Ping request syntax,
- **Send test message** – A message with a payload of "1" will be sent to "validation" topic.



The **MQTT Last Will and Testament (LWT)** section is used to configure the mechanism for notifying MQTT brokers of unexpected device disconnection.

- **LWT** – Enables or disables Last Will and Testament. If the option is active, the MQTT broker will receive the preprogrammed message in the event of a connection loss by the device,

- **QoS (Quality of Service)** – Defines the level of service quality for the LWT message:
  - **QOS0** – The message is sent once, with no confirmation of delivery,
  - **QOS1** – The message is sent repeatedly until the broker confirms reception,
  - **QOS2** – The message is sent and confirmed in a way that ensures a single delivery.
- **LWT retain** – If enabled, the LWT message will be saved on the broker an will be available for new topic subscribers,
- **LWT Topic** – Defines MQTT topic, to which the LWT message will be published,
  - Example: `/device/MAC_address/lwt`.
- **LWT Message** – Content of the message the broker will publish when the device unexpectedly looses connection,
  - For example it may be `offline` or any other announcement specifying device status.



- **SSL/TLS** – Enable connection encryption using SSL/TLS protocols,
- **SSL certificate mode** – Certificate support mode:
  - **Use Certificate Bundle** – Use the built-in certificate set,
  - **Use Uploaded Certificate** – Use the uploaded certificate set,
  - **Disable SSL verification**.
- **Skip cert CN check** – Skip verification of the Common Name (CN) field in the certificate. Useful in test or non-standard environments,
- **Use Client certificate** – Enable using a client certificate for authentication on the server side,
- **Client key password** – Client key password (if required).



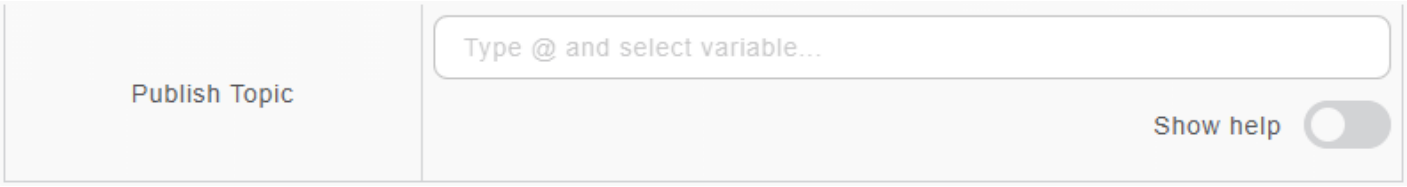
- **SSL server root certificate** – Option to upload a root server certificate in order to verify encrypted connections,
- **Client certificate** – Option to upload a client certificate used for authenticating the device by the MQTT broker,
- **Client key** – Option to upload a private key for the client certificate.

Publish topic

After the MQTT protocol is configured, triggers causing data to be sent to a defined topic (**Publish Topic**) should be declared. Similarly to HTTP Client, there are three triggers available:

- **On-new** – Send the MQTT message every time an RFID card is applied to the reader,
- **On-hold** – Send a MQTT message if a card remains in the read field for at least 5 seconds (default **hold time**),
- **On-remove** – Send a MQTT message if the RFID tag is removed from the read area.

Each of the triggers operates independently, which allows configuring their operation to suit the user's needs. The user can choose what events cause a MQTT message to be sent.



For publish topic content:

```
uid=%uid%&name1=%usr_name1%&name2=%usr_name2%
```

Data visible on the server side:

```
uid=87878A4C&name1=John&name2=Smith
```

Subscribe topic

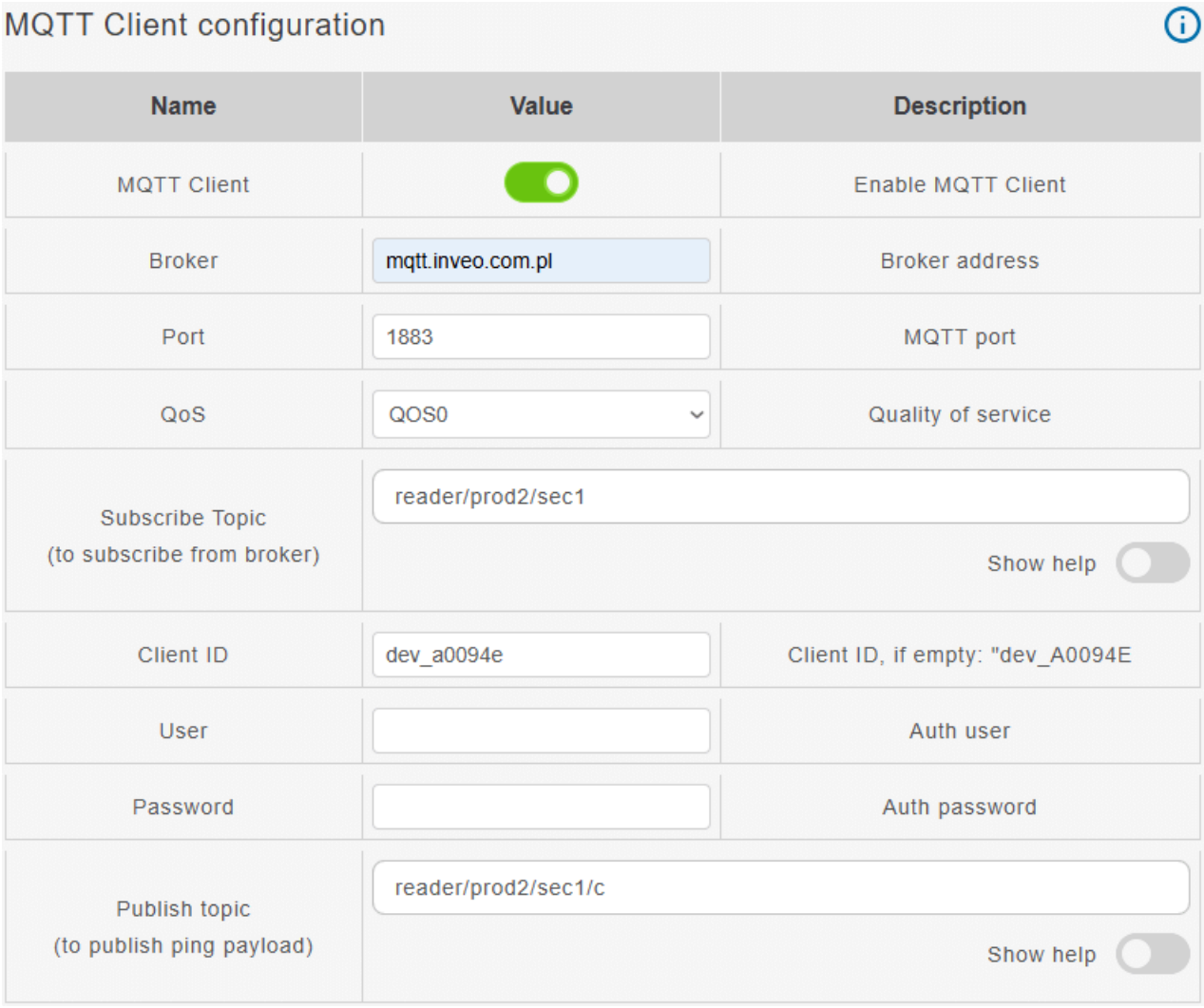
The device can be controlled via the MQTT protocol. To make this possible, **Subscribe Topic** from which the device will receive control commands must be configured. To do that, enter the required MQTT broker settings on the **Services / MQTT** site and define the **Subscribe Topic**.

> **Warning**
>
> **Special attention should be given to the correct order in which the commands are sent. For example: sending `lcd_text[0-3]` and then `lcd_clr` will clear the display. Correct order: `lcd_clr`, then `lcd_text[0-3]`**

Command	Value	Description
out_run	INTEGER	Activate output in default mode
out_on	INTEGER	Output activation. eg. `"out_on":1` activates output No. 1
out_off	INTEGER	Output deactivation
txt0	STRING	Internal variable for controlling the display
txt1	STRING	As above

txt2	STRING	As above
txt3	STRING	As above
template_index	INTEGER	Template index to be displayed
template_name	STRING	Template name to be displayed
buzz	INTEGER	Sound emission: 1 – accept, 2 – reject, 3 - short signal example: `"buzz":2` - buzzer will emit the card rejection sound signal
userdb_ver	STRING	Returns the card database stored in RAM (if the base is sent to the reader via HTTP Client). If the card database is updated dynamically or dependent on specified conditions, this command allows to monitor its version and check if changes have been made
userdb_store	INTEGER	Transcription of card data from **RAM** to **FLASH** (**1**). This operation should be performed only after all user modifications have been made, and as rarely as possible. It should be noted that the amount of guaranteed memory write cycles is 10 000
cardlog_remove_n	INTEGER	Delete last x card logs
cardlog_remove_all	BOOLEAN	Delete all card logs
enter_state	**FLAG_OPT=1** **FLAG_IN=4** **FLAG_OUT=8** **FLAG_EXT=10** **FLAG_KBD=20** **FLAG_QR=40** ***(HEX)***	**Setting of entry mode. Returns the value in decimal format.** **FLAG_OPT (0d1, 0x01):** Optional condition (e.g. private/work) **FLAG_IN (0d4, 0x04):** Sets the mode to entrance – records the entrance time. **FLAG_OUT (0d8, 0x08):** Sets the exit mode – records the exit time. **FLAG_EXT (0d16, 0x10):** Card application to an external reader. **FLAG_KBD (0d32, 0x20):** Keypad record *(Clocker CL-30 only)* **FLAG_QR (0d64, 0x40):** Built-in QR code reader record *(option)*. **Example:** The setting for **optional entrance** has a value of **5**, which is the result of combining **FLAG_OPT (0d1)** with **FLAG_IN (0d4)**
rfid_en	BOOLEAN	Enable (**1**) or disable (**0**) card reading

Example configuration:



Information

The device sends data to the `reader/prod2/sec1` topic,
It receives the messages from `reader/prod2/sec1/c`

Tip

The device supports all HTTP protocol commands. They should be divided with "&".

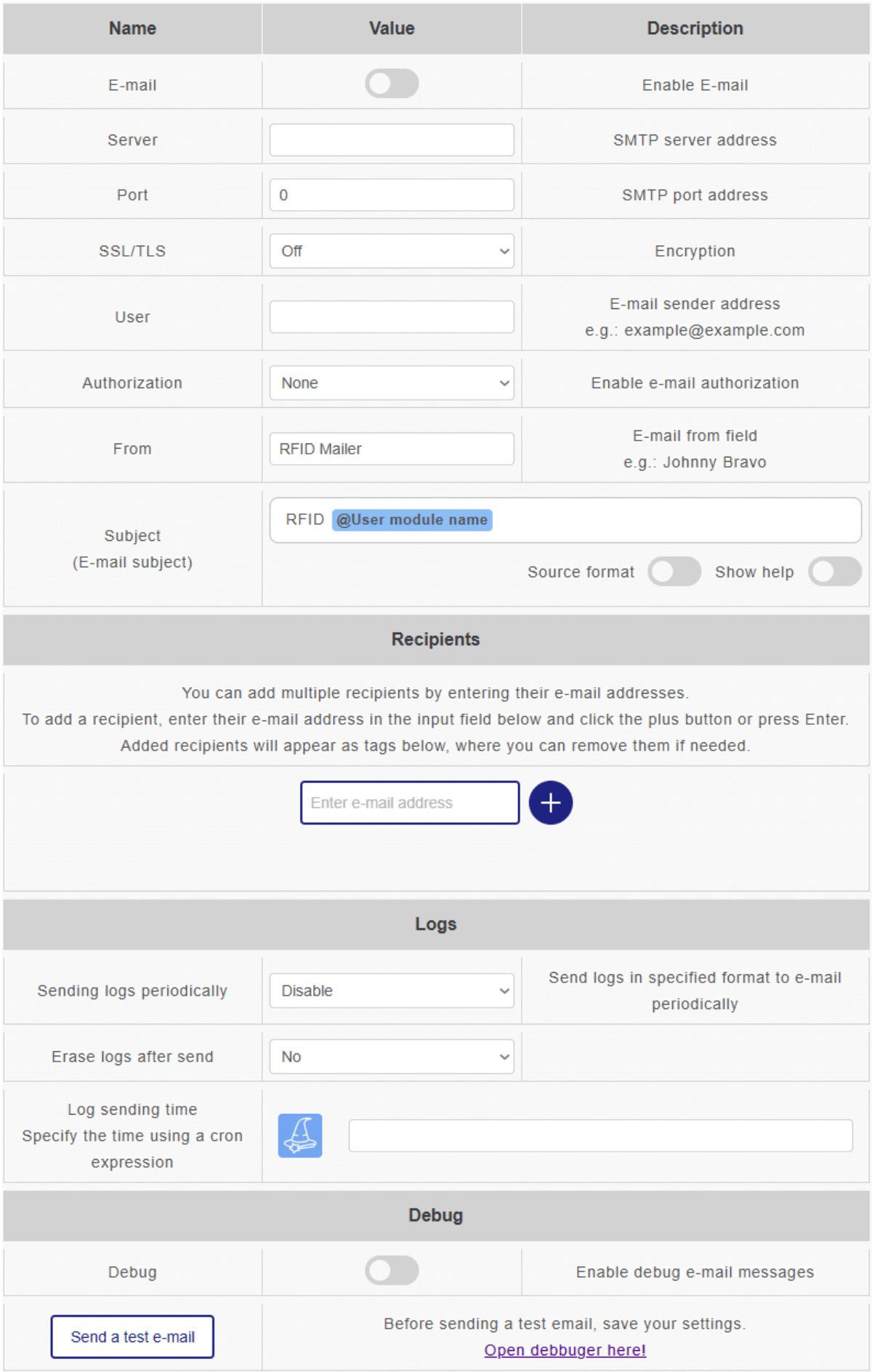
Example message sent to `reader/prod2/sec1/c` :

```
lcd_text1= Place an RFID tag&buzz=3
```

The device will display **Place an RFID tag** in the second LCD line and emit a short (**beep**). It's possible to use the same topic for publishing and subscribing – this enables instant device control after data is sent to **Publish Topic**.

E-mail

The device can send encrypted e-mail messages that can be configured in the **Services** tab.



- **E-mail** – Activate the SMTP service,
- **Server** – SMTP server address,

- **Port** – SMTP server port, on which messages are received,
- **SSL/TLS** – Encryption configuration:
  - **Off** – No encryption, connection unencrypted,
  - **SSL/TLS** – SSL/TLS Encryption,
  - **STARTTLS** – Enables the use of STARTTLS during connection.
- **User** – User name used for authentication in the SMTP server,
- **Authorization** type selection: none or password,
- **Password**,
- **From** – "From" field visible in the e-mail message header,
- **Subject** – Subject of the e-mail message and name of the attachment file (eg. logs),
- **Recipients** – List of message recipients, e-mail addresses separated by commas,
- **Sending logs periodically** – Log file extension to be sent as an attachment:
  - **Disable** – Log sending disabled,
  - **CSV**,
  - **XML**,
  - **JSON**,
  - **Text**.
- **Erase logs after send** – Option to erase the logs after they are successfully sent,
- **Time** – CRON format log sending schedule:
  - **Syntax:** seconds, minutes, hours, day, month, day of the week.

Example

`0 0 0 ? * *` **– every day at midnight**

- **Debug** – Debug mode mode allowing for diagnostics of SMTP configuration errors.

Configuration testing

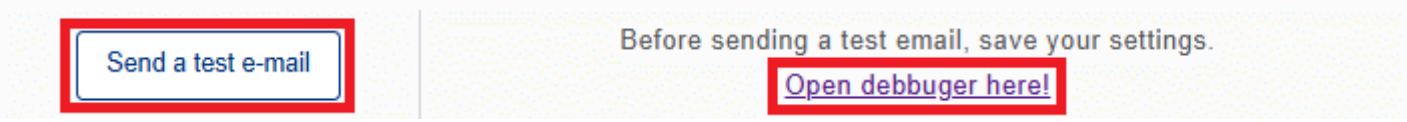
After applying the settings, the user can test the configuration using the **Send a test e-mail** button. The device will attempt to send a test message, that should reach the recipient immediately. If no message is received, check the SPAM folder.

Troubleshooting

If problems with message sending arise, **Debug** mode can be enabled in order to diagnose the errors:

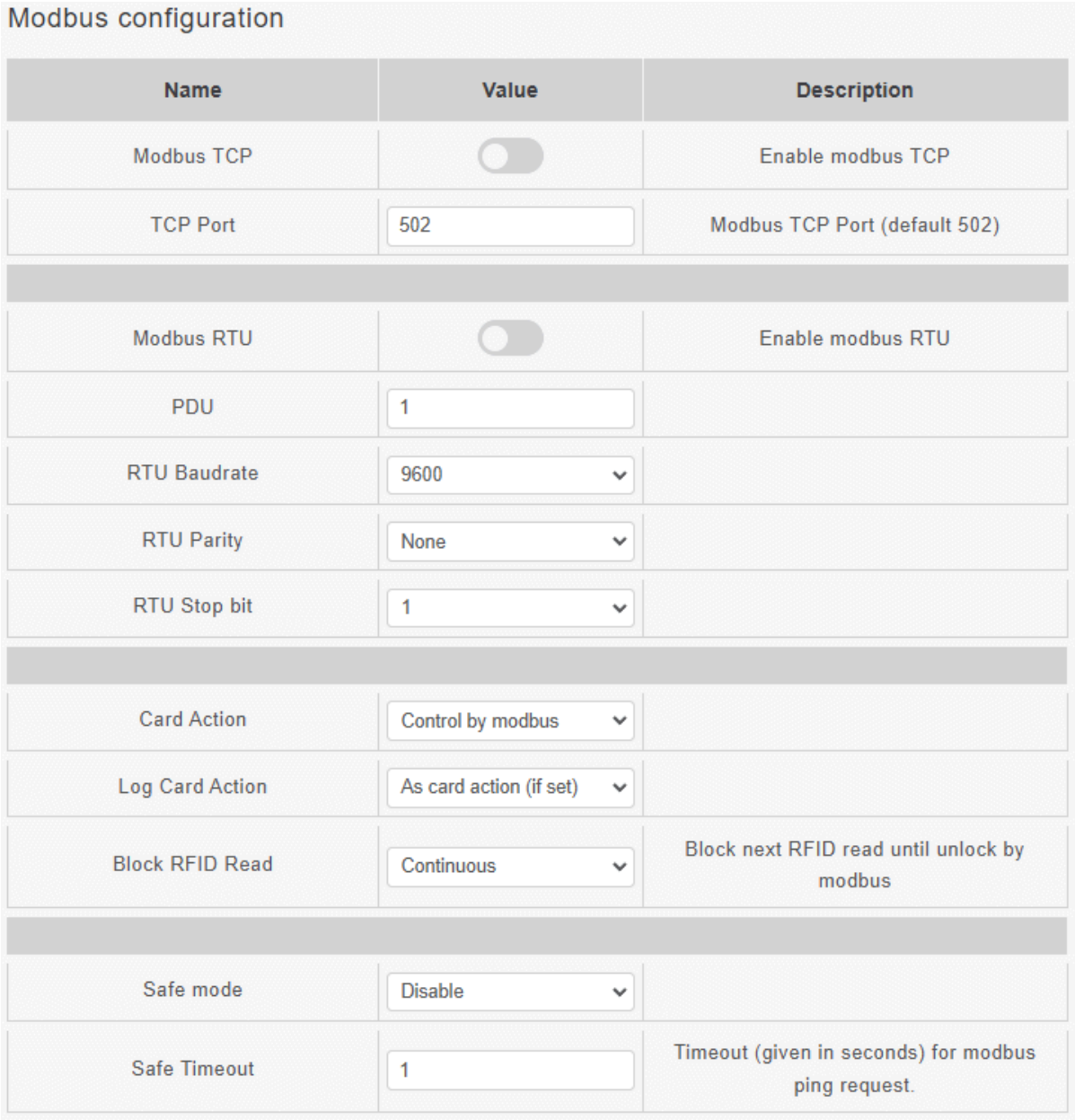
1. Open the debugging page in a new browser tab, typing in:
   `http://<IP_address>/protect/dmesg.html`
   or click **Open debugger here!** on the configuration page,
2. Push the **Send a test e-mail** button,

3. Details concerning the message sending problems will appear in the debugger's window. Analysis of this information will allow the problem to be identified and solved.



Modbus TCP/RTU

The device supports the Modbus TCP and Modbus RTU protocols. Connection configuration is available on the webpage under the **Services / Modbus** tab.



- **Modbus TCP** – Enable Modbus TCP service,
  - **TCP port** – Modbus TCP protocol port (**default 502**),
- **Modbus RTU** – Enable Modbus RTU service,
  - **PDU** – Device Modbus address,
  - **RTU Baudrate** – Data transfer speed; available values:
    - 1200
    - 2400
    - 4800
    - 9600

        - 19200
        - 38400
        - 57600
        - 115200
    - **RTU Parity** – Parity settings:
        - None
        - Odd
        - Even
    - **RTU Stop bit** – Number of stop bits:
        - 1
        - 2
- **Card action** – Reaction of the reader for card application:
    - **Control by modbus** – Control of the device by Modbus only,
    - **Reader card action** – Reaction compliant with settings from the **Card Action** tab.
- **Log Card Action** – Logging of events during a correct connection with master (only for the **Card Action** → **Reader card action** setting):
    - **As card action (if set)** – Logging compliant with **Card Action** settings,
    - **Omit log action** – Omit the **Card Action** settings.
- **Block RFID read** – Block read of a next tag:
    - **Block** – After a tag is read, value 1 is sent to the Holding Register 1. Readout of another transponder is possible after a read flag is reset (value 0 in Holding Register 1),
    - **Continous** – Automatic erasure of the read flag – allows instantaneous reading of other RFID tags.
- **Safe mode** – Reaction of the reader in the event that Modbus connection is lost:
    - **Disable**,
    - **No action** – No reader reaction for an applied tag,
    - **Standalone (Reader Card Action)** – Transition to autonomous mode.
- **Safe Timeout** – Time until the reader enters **Safe mode**.

The device supports the following Modbus functions:

- 0x01 Read Coils
- 0x03 Read Holding Register
- 0x05 Write Single Coil
- 0x06 Write Single Register
- 0x0F Write Multiple Coils
- 0x10 Write Multiple Registers

Holding Registers addresses

Address	R/W	Description
1	R/W	Read flag Read: 1 – a new transponder has been read Write: 0 – zeroizing of read flag
2	R	ID_LEN
3	R	Card ID/UID/CSN [0]
4	R	Card ID/UID/CSN [1]
5	R	Card ID/UID/CSN [2]
6	R	Card ID/UID/CSN [3]
7	R	Card ID/UID/CSN [4]
8	R	Card ID/UID/CSN [5]
9	R	Card ID/UID/CSN [6]
10	R	Card ID/UID/CSN [7]
11	R	Card ID/UID/CSN [8]
12	R	Card ID/UID/CSN [9]
13	R	Reserve
14	R	User ID – **65535** indicates an unknown tag
20	R/W	Output #0 control according to **I/O Settings** tab
21	W	Output #0 control in bi-stable mode
22	W	Output #0 enable for a time set as a parameter (value * 0,1s)
23	W	blink_on time (value * 0,1s)
24	W	blink_off time (value * 0,1s)

25	W	Output #0 control in flashing mode (time set in registers 23/24)
26	R/W	Output #1 control according to **I/O Settings** tab
27	W	Output #1 control in bi-stable mode
28	W	Output #1 enable for a time set as a parameter (value * 0,1s)
29	W	blink_on time (value * 0,1s)
30	W	blink_off time (value * 0,1s)
31	W	Output #1 control in flashing mode (time set in registers 29/30)

Logs

Address	R/W	Description
1200	R/W	Log amount [MSB], sending 0 will delete all logs
1201	R	Log amount [LSB]
1202	R	Selected log [MSB]
1203	R	Selected log [LSB] (log numbering starts from 0)
1204	W	Delete oldest logs, amount set as a parameter
1205	R	Timestamp [MSB]
1206	R	Timestamp [LSB]
1207	R	Reserve
1208	R	Log source (main reader, auxiliary reader)
1209	R	Action
1210	R	idx – **FFFF** indicates an unknown tag
1211	R	UID – Code saved in ASCII
...1231	R	Last register with optional data

Display

Address	R/W	Description
2000	R/W	LCD Mode: 0 – autonomous operation -1 – Modbus control Value from 1 to 65534 – text display time in seconds. After the time passes, the display will enter autonomous mode
2001	R/W	LCD TIME, if LCD Mode=1: 0 – do not display time on LCD 1 – display time on LCD
2002	R/W	LCD Clear: 1 – clear LCD
2003	W	Display backlight: 0 – disable backlight 1 – enable backlight >1 – backlight time (value * 0.1 sec)
2010-2029	W	LCD first line – UTF-8 encoding
2030-2049	W	LCD second line – UTF-8 encoding
2050-2069	W	LCD third line – UTF-8 encoding
2070-2089	W	LCD fourth line – UTF-8 encoding

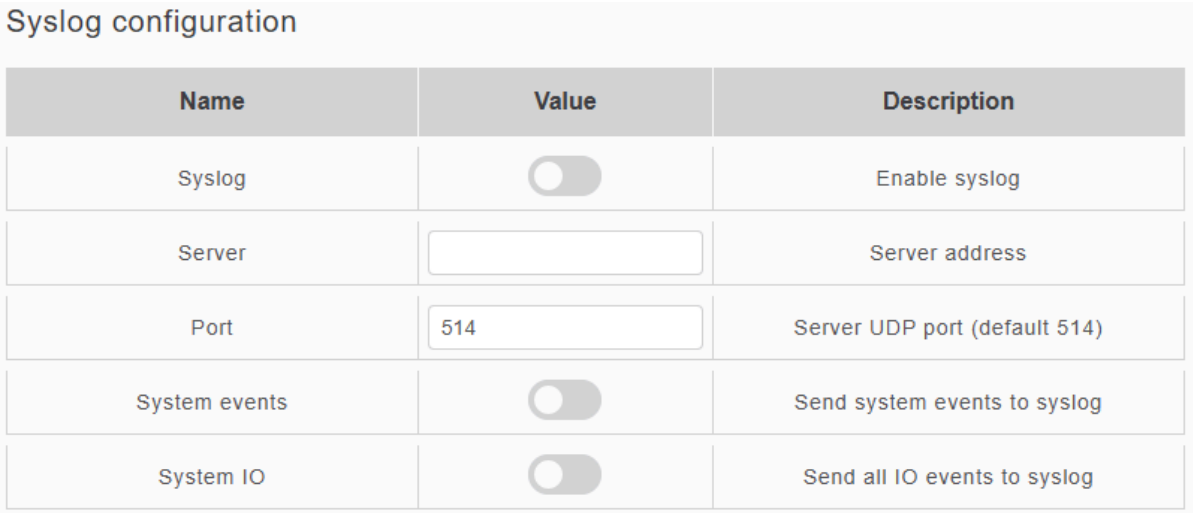
Single Coil addresses

Address	R/W	Description
1	R/W	Output #0 control according to **I/O Settings**
2	R/W	Output #1 control according to **I/O Settings**
3	R	Output #0 coil state
4	R	Output #1 coil state
11	R	Input #0 state
12	R	Input #1 state

13	R/W	Red LED control
14	R/W	Green LED control
15	W	Sound generator control – ACCEPT sound
16	W	Sound generator control – REJECT sound
17		Reserve
18	R/W	Reset status Read: 1 – a restart happened (eg. because of power loss)

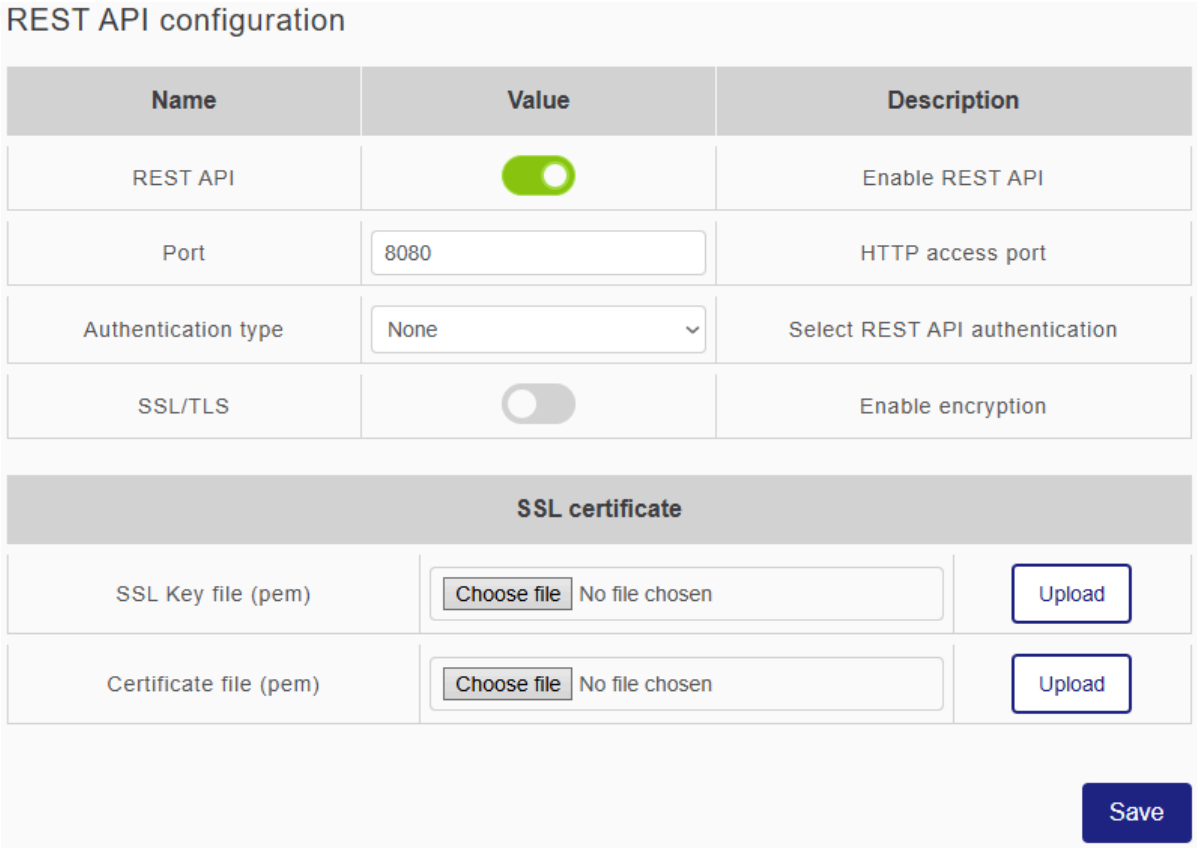
Syslog

The device can forward data to a Syslog server. The user can choose what events will be sent. Configuration takes place in the **Syslog Settings** table located in the **Services/Syslog** tab.



- **Syslog** – Enable syslog client,
- **Server** – Syslog server address,
- **Port** – Port number, the server is listening on – **default 514**,
- **System events** – Send system events to the syslog server,
- **System IO** – Send input/output events to the syslog server.

REST API



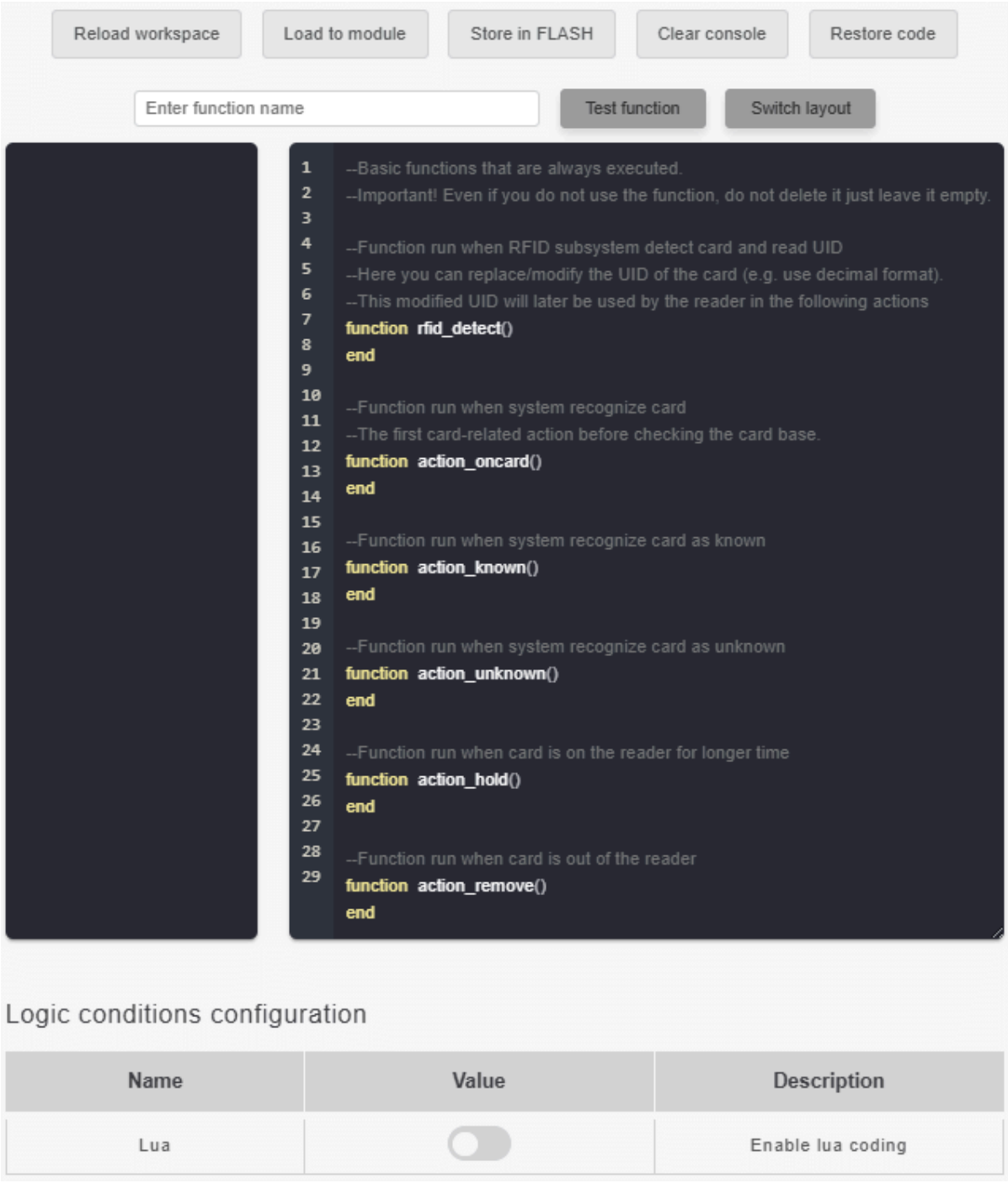
- **REST API** – Enable REST API,
- **Port**,
- **Authentication type**:
  - **None**,
  - **Basic authentication** – User name and password authentication,
  - **API KEY** – API key authentication, utilizing a key generated in **Administration/Access**. The key should be entered as a parameter, preceded with `apikey=`.
- **User** – User name used with Basic authentication,
- **Password** – Password used with Basic authentication,
- **SSL/TLS** – Enable/disable encryption,
- **SSL Key file (pem)** – Allows upload of SSL key (pem format),
- **Certificate file (pem)** – Allows upload of certificate file (pem format).

Example

API key value: `12345678901234567890`
REST API request syntax: `http://<IP_address>:port/io/outputs/1?apikey=12345678901234567890`

Lua Scripts

This tab allows an advanced user to define custom actions and RFID card control logic using the **Lua** script language, effectively customising the reader's firmware per specific needs.



Interface description

The main tab section contains a code editor that can be used to create and edit Lua scripts. Those scripts allow customization of device operation depending on different RFID card events.

Control buttons

- **Reload workspace** – Reload the editor,
- **Load to module** – Upload the Lua script to the device (without saving to non-volatile memory),

- **Store in FLASH** – Save the Lua script to device memory. The script will remain active after a restart,
- **Clear console** – Clear the output console,
- **Restore code** – Restore the default Lua code.

> **Tip**
>
> **The "Lua" option located at the bottom of the tab allows the user to enable or disable Lua script support in the device.**

Available system functions

Lua functions can be used to make the device react to different events.

System functions

- `start()` – Recalled after the system start-up,
- `timeout()` – Recalled after `set_timeout(ms)` time has passed.

RFID functions

- `rfid_detect()` – Recalled when the system detects an RFID card and reads its UID,
- `action_begin()` – If the **Wait for server reply** function is active, this function will be recalled before `action_known()` and `action_unknown()` functions,
- `action_request()` – If the **Wait for server reply** function is active, this function will be recalled before the request is sent to the server (e.g. for IO control while awaiting server response),
- `action_known()` – Recalled when the read card can be found in the card database or when **Action by HTTP status code** is enabled and the reader receives a 200 response from the server,
- `action_unknown()` – Recalled when the read card can't be found in the card database or when **Action by HTTP status code** is enabled and the reader receives a 401 or 404 response from the server,
- `action_hold()` – Recalled when a card remains in the read area for a longer time,
- `action_remove()` – Recalled when a card is removed from the read area.

I/O functions

- `on_key_pressed(key)` – Recalled after a keypad button is pressed. `key` is the key index,
- `in0_chg()` – Recalled when the input No. 0 state changes,
- `in1_chg()` – Recalled when the input No. 1 state changes,
- `out0_chg()` – Recalled when the output No. 0 state changes,
- `out1_chg()` – Recalled when the output No. 1 state changes.

> **Warning**
>
> **Changing the output state inside of the function will call it again, which can introduce a program loop.**

HTTP functions

- `httpc_client_rcv(code, body, content_type)` – Recalled after the reader receives a response from the server. The received arguments can be used in the script elsewhere:
  - `code` – Response code,
  - `body`,
  - `content_type`:
    - `0` – plain,
    - `1` – JSON,
    - `2` – XML.
- `httpc_client_err(code)` – Recalled after a request sent with `dev.http()` encounters a transport or client error (e.g. no server, timeout etc.). The function is **not** called if the reader receives a 4xx/5xx response:
  - `code` – STRING.

Lua libraries

The system offers several libraries that enable interaction with the device. The `card`, `mf`, `rfid`, `userdb` and `dev` libraries are specific to Inveo devices. All other libraries are standard.

`card` LIBRARY

- `card.getFlag()` – Returns flags associated with the read RFID card,
- `card.getUserId()` – Returns the user ID associated with the read RFID card,
- `card.getUserName()` – Returns both user name fields (**name1** and **name2**),
- `card.getParentFlag()` – Returns the parent flag for the read child card. For a parent card, returns its own flag,
- `card.getParentUserId()` – Returns the parent ID for the read child card. For a parent card, returns its own ID,
- `card.getParentUserName()` – Returns the parent name for the read child card. For a parent card, returns its own name.

`mf` (MIFARE CLASSIC) LIBRARY

- `mf.auth_ee(key_no, block, ab)` – Authorization using a key from a secure memory. Returns `0` if the operation has failed:
  - `key_no` – Key index in device's memory,
  - `block` – Mifare block index,
  - `ab` – `0` - Key A, `1` - Key B.
- `mf.auth_key(key, block, ab)` – Authorization using a provided key:
  - `key` – HEX key in ASCII form (e.g. `"FFFFFFFFFFFF"`),
  - `block` – Mifare block index,
  - `ab` – `0` - Key A, `1` - Key B.
- `mf.rd(block)` – Mifare Classic block read operation. Returns the data length and contents. A length of `16` indicates a successful operation,
- `mf.wr(block, data)` – Mifare Classic block write operation (has to be exactly 16 bytes),
- `mf.deauth()` – De-authorization.

`rfid` LIBRARY

- `rfid.getUidBin()` – Returns the UID as binary data,
- `rfid.getUidBinLen()` – Returns the UID length in bytes,
- `rfid.getUidAscii()` – Returns the UID as ASCII (HEX),
- `rfid.getUidAsciiLen()` – Returns the UID length in ASCII,
- `rfid.overwriteUidAscii(uid)` – Overwrites the UID used by the system,
- `rfid.pause(ms)` – Pauses processing of RFID for a given time (ms).

`userdb` LIBRARY

- `userdb.get(id)` – Returns the `id` record data: `name1`, `name2`, `uid`, `flags`, `parent` or `nil` if a card does not exist,

- `userdb.find_uid(uid)` – Finds a card by its ASCII `uid`. Returns the `id` or `nil`,
- `userdb.find(start [, uid] [, name])` – Finds the next card from the `start` index. The `uid` and `name` fields are optional. Returns the `id` or `nil`,
- `userdb.info()` – Returns the database statistics:
  - `ver` – Database version,
  - `dirty` – Base is not stored in FLASH,
  - `n_users` – Amount of users in the database,
  - `n_cards` – Amount of cards in the database.
- `userdb.add_user(n1, n2, uid [, flags] [, idx])` – Adds a new parent user to the card database:
  - `n1` – `name1` field,
  - `n2` – `name2` field,
  - `uid` – Card UID,
  - `flags` – Optional `active` user flag. Default `flags` = `1` (`active`),
  - `idx` – Optional slot index. Default `idx` = `-1` (first vacant slot).
- `userdb.add_uid(parent_id, uid)` – Adds a new child card `uid` to the card database and assigns it to the `parent_id` parent user,
- `userdb.remove_user(id)` – Removes the user (`id`) and all child cards,
- `userdb.remoce_card(uid)` – Deletes a single card by its ASCII `uid`,
- `userdb.save()` – Database write request,
- `userdb.reload()` – Database reload request.

`dev` LIBRARY

- `dev.set_timeout(ms)` – Sets the internal timer to a value in ms and recalls the `timeout()` function after it's exhausted,
- `dev.clear_timeout()` – Clears and deactivates the timer,
- `dev.delayms(ms)` – Blocks the thread for `ms`. Counts to the 1s limit for function execution.

I/O

- `dev.out_on(ch)` – Activates the output on a given channel `ch`,
- `dev.out_off(ch)` – Deactivates the output on a given channel `ch`,
- `dev.out_single(ch, time)` – Sets a single pulse, wit the length of `time` (in 0,1s) on the output channel `ch`,
- `dev.out_blink(ch, ton, toff, cnt)` – Blinks the output `ch` `cnt` - times with pulses of `ton` length separeted by `toff` (in 0,1s),
- `dev.out_get(ch)` – Returns the `ch` output state,
- `dev.in_get(ch)` – Returns the `ch` input state,
- `dev.in_get_cnt(ch)` – Returns the `ch` input counter,
- `dev.get_enter_state(val)` – Returns the function button state, e.g. 0x04 – ENTER_FLAG_IN - entry (consult column 8 of the **Events preview** table) *(Clocker CL-20 only)*,

- `dev.set_enter_state()` – Sets the function button state and their backlight e.g. 0x04 – ENTER_FLAG_IN - entry (consult column 8 of the **Events preview** table) *(Clocker CL-20 only)*.

Signaling

- `dev.buzz(sound)` – Buzzer control (`sound` : 0 - accept, 1 - reject, 2 - pulse),
- `dev.lcd_clear()` – Clears the LCD screen.
- `dev.lcd_text(txt)` – Displays text on the screen,
- `dev.lcd_text_line(line, txt)` – Displays `txt` in the specified display `line` ,
- `dev.lcd_text_timeout(s)` – Sets the time in seconds, after which the display returns to displaying the default text,
- `dev.lcd_template_idx(idx)` – Displays the template with an index of `idx` on the display *(Proxer only)*,
- `dev.lcd_template_name(name)` – Displays the template with a name of `name` on the display *(Proxer only)*,
- `dev.var_txt_set(idx, txt)` – Sets `txt` as the value of the `idx` dynamic variable,
- `dev.var_txt_get(idx)` – Recalls the `idx` dynamic vatiable.

RTC SRAM

The embedded RTC provides 32 bytes of battery-backed SRAM memory, allowing important data to be retained in the case of a reader power failure.

- `dev.sram_write(offset, data)` – Writes `data` (STRING) to the specified `offset` (32-63). Returns:
  - `0` – OK,
  - `-1` – `offset` out of range,
  - `-3` – Error (contact service).
- `dev.sram_read(offset, len)` – Read the `len` amount of data from the specified `offset` (32-63). Returns:
  - `-1` – Offset out of range,
  - `-2` – `len` = 0 or `offset + len` > 64,
  - `-3` – Error (contact service).

Device identification

- `dev.get_mac_ascii()` – Returns the Ethernet interface's MAC address as ACII text,
- `dev.get_ip_ascii()` – Returns the device's IP address as ASCII text,
- `dev.mod_name()` – Returns the device's model name,
- `dev.name()` – Returns the user-defined devicee name,
- `dev.get_version()` – Returns the firmware revision,
- `dev.isScanEn()` – Returns the QR code scanner *(option)* active flag from the configuration.

Logging and debugging

- `dev.log(msg)` – Saves `msg` to the device's log,
- `dev.console(msg)` – Dispays `msg` in the Lua console.

Communication

E-mail

- `dev.email(recipient, subject, text)` – Sends an e-mail:
  - `recipient` – Recipient's address,
  - `subject` – Message topic,
  - `text` – Message content.

HTTP

- `dev.http(url, data, data_length, method, content_type [, headers])` – Sends a HTTP request:
  - `url` – Address, the device will send the request to. Entering `nil` here will send the request to the address specified in the **HTTP Client** tab,
  - `data`,
  - `data_length`,
  - `method`:
    - `0` - GET,
    - `1` - POST,
    - `2` - PUT,
    - `3` - DELETE.
  - `content_type`:
    - `0` - plain,
    - `1` - JSON,
    - `2` - XML.
  - `headers` – Optional headers table `{["key"]="value", ...}`. Sending an empty STRING (`""`) deletes the header.

Example

```
dev.http(nil, '{"uid":"AABBCCDD"}', 20, 1, 1)  -- POST JSON, URL from config

dev.http(nil, body, #body, 1, 1, {
["Authorization"] = "Bearer TOKEN",
["Accept"] = "application/json",
})

dev.http(nil, body, #body, 1, 1, {
["Authorization"] = "",  -- disables the setup auth
})

function httpc_client_err(code)
-- e.g. code == "-1"  (no server / perform error)
end
```

HTTP Server – Request handling

- `dev.httpdscript(name, args)` – Function recalled when an HTTP request begins with `/data/script/`. Can be used to trigger a specific Lua script in response to a request sent to the reader,
- `dev.httpd_resp(text)` – Sets the HTTP response.

Example

```
function httpdscript(name, arg)
    dev.console("name:"..name.." arg:"..arg)

    local v1,v2,v3 = arg:match("v1=(%d+)&v2=(%d+)&v3=(%d+)")
    dev.console("parsed arg:"..v1.." "..v2.." "..v3)

    if name == "test.json" then
        dev.httpd_resp('{"result":"OK"}')
    elseif name == "test.xml" then
        dev.httpd_resp('<AastraIPPhoneStatus Beep="yes"><Session/><Message type="alert"
Color="darkgreen" Timeout="3">Door open</Message></AastraIPPhoneStatus>')
    else
        dev.httpd_resp("")
    end
end
```

MQTT

- `dev.mqtt(topic, data, length)` – Publishes an MQTT message:
  - `topic`,
  - `data`,
  - `data_length`,
  - `retain` – Retain flag (BOOLEAN).

`table` LIBRARY

- `table.insert(t, [pos], value)` – Inserts `value` to array `t`. If `pos` is specified, inserts the element to a specified position, shifting other elements,
- `table.remove(t, [pos])` – Removes the element from `pos` in `t` array. If `pos` is not specified, removes the last added element,
- `table.sort(t, [, comp])` – Sorts the `t` array in ascending order. Optionally, the `comp(a, b)` comparison function can be added; it should return `true` if `a` should be before `b`,
- `table.concat(t [, sep [, i [, j]]])` – Combines the `t` array contents into a single string, separating them with `sep` from index `i` to `j`,
- `table.pack(...)` – Returns an array containing all function arguments. Remembers the amount of arguments as `n`,
- `table.unpack(t [, i[, j]])` – Returns the contents of `t` array as separate values for indexes `i` to `j`. Default: `i = 1`, `j = #t`.

`string` LIBRARY

- `string.bytebyte(s [, i [, j]])` – Returns the byte code for `s` characters, from `i` to `j`,
- `string.char(...)` – Creates a string from the specified values,
- `string.len(s)` – Returns the character string length,
- `string.lower(s)` – Returns `s` with upper-case letters converted to lower-case,

- `string.upper(s)` – Returns `s` with lower-case letters converted do upper-case,
- `string.sub(s, i [, j])` – Returns the `s` substring from `i` to `j`,
- `string.rep(s, n)` – Returns `s` repeated `n` times,
- `string.reverse(s)` – Returns the inverted text of `s`,
- `string.find(s, pattern [, i [, plain]])` – Searches for `pattern` in `s`, returns the indexes or `nil`. Setting `plain` to `true` searches with no patterns,
- `string.match(s, pattern [, i])` – Returns the first pattern match,
- `string.gsub(s, pattern, repl [, n])` – Substitutes `pattern` with the `repl` string a maximum of `n` times. Returns the new string and the amount of substitutions.

`os` LIBRARY

- `os.time([table])` – Returns current time in UNIX format or converts `table` to seconds,
- `os.date([format [, time]])` – Returns the date/time in the form of formatted text or a table,
- `os.clock()` – Returns the time spent for program exacution by the processor,
- `os.difftime(t1, t2)` – Returs the difference in seconds between `t1` and `t2`,

`math` LIBRARY

- `math.abs(x)` – Absolute value of `x`,
- `math.ceil(x)` – Round up to the nearest whole number,
- `math.floor(x)` – Round down,
- `math.fmod(x, y)` – Remainder of the `x` by `y` division operation,
- `math.max(x, ...)` – The biggest number specified,
- `math.min(x, ...)` – The smallest number specified,
- `math.sqrt(x)` – Square root of `x`,
- `math.exp(x)` – `e` to the power of `x`,
- `math.log(x [, base])` – Natural logarithm `ln(x)` or `log_b(x)` (if `base` is specified),
- `math.pow(x, y)` – Exponentiation of `x^y`,
- `math.modf(x)` – Splits `x` into integer and fractional parts `int, frac`,
- `math.random([m [, n]])` – Random number between 0 and 1,
- `math.randomseed(seed)` – Sets a seed of the random number generator,
- `math.deg(x)` – Converts radians to degrees,
- `math.rad(x)` – Converts degrees to radiants,
- `math.sin(x)` – Sinus (`x` in radians),
- `math.cos(x)` – Cosinus (`x` in radians),
- `math.tan(x)` – Tangens (`x` in radians),
- `math.asin(x)` – Arcus sinus,
- `math.acos(x)` – Arcus cosinus,
- `math.atan(x)` – Arcus tangens,

- `math.atan2(y, x)` – Arcus tangens `y/x` including the quadrant.

`utf8` LIBRARY

- `utf8.len(s [, i [, j]])` – Returns the number of UTF-8 characters in the substring `s` from `i` to `j`,
- `utf8.offset(s, n [, i])` – Returns the byte index in `s` of the beggining of the nth character (if `n` positive) or the nth from the end (if `n` negative),
- `utf8.codepoint(s [, i [, j]])` – Returns at least one Unicode point code of characters from `s` between `i` and `j`,
- `utf8.char(...)` – Creates an UTF-8 string from the given Unicode point codes,
- `utf8.codes(s)` – Returns the iterator, that returns subsequent byte indexes and Unicode point codes in `s`,
- `utf8.charpattern` – Lua pattern for matching a single UTF-8 character.

`pack` LIBRARY

- `pack.pack` – Packs data to a binary string,
- `pack.unpack` – Unpacks data from a binary string,
- `pack.packsize` – Returns the size of a packet.

Application examples

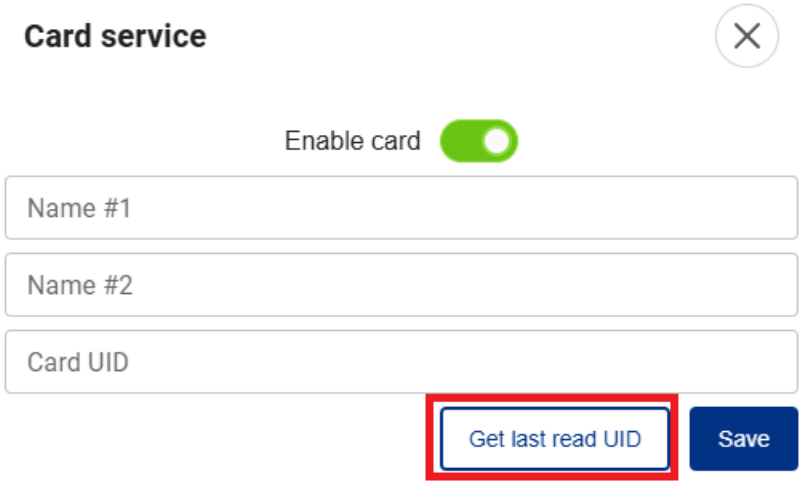
Example 1: Room access control

Objective: Opening doors with authorized RFID cards.

Configuration:

1. Adding users and RFID cards

- Go to the **Card database** tab in the device interface.
- **Add a new RFID card to database** and enter a user name.
- Apply the RFID card to the reader - press **Get last read UID**.
- **Save changes**. Repeat for each user.



2. Relay output configuration

- Go to the **IO Settings** tab.
- In the **Out #0 Mode** section, set mode to **On Period**.
- In the **Out #0 Time** section, set the relay activation time (eg. 20 = 2 seconds).
- **Save changes**.

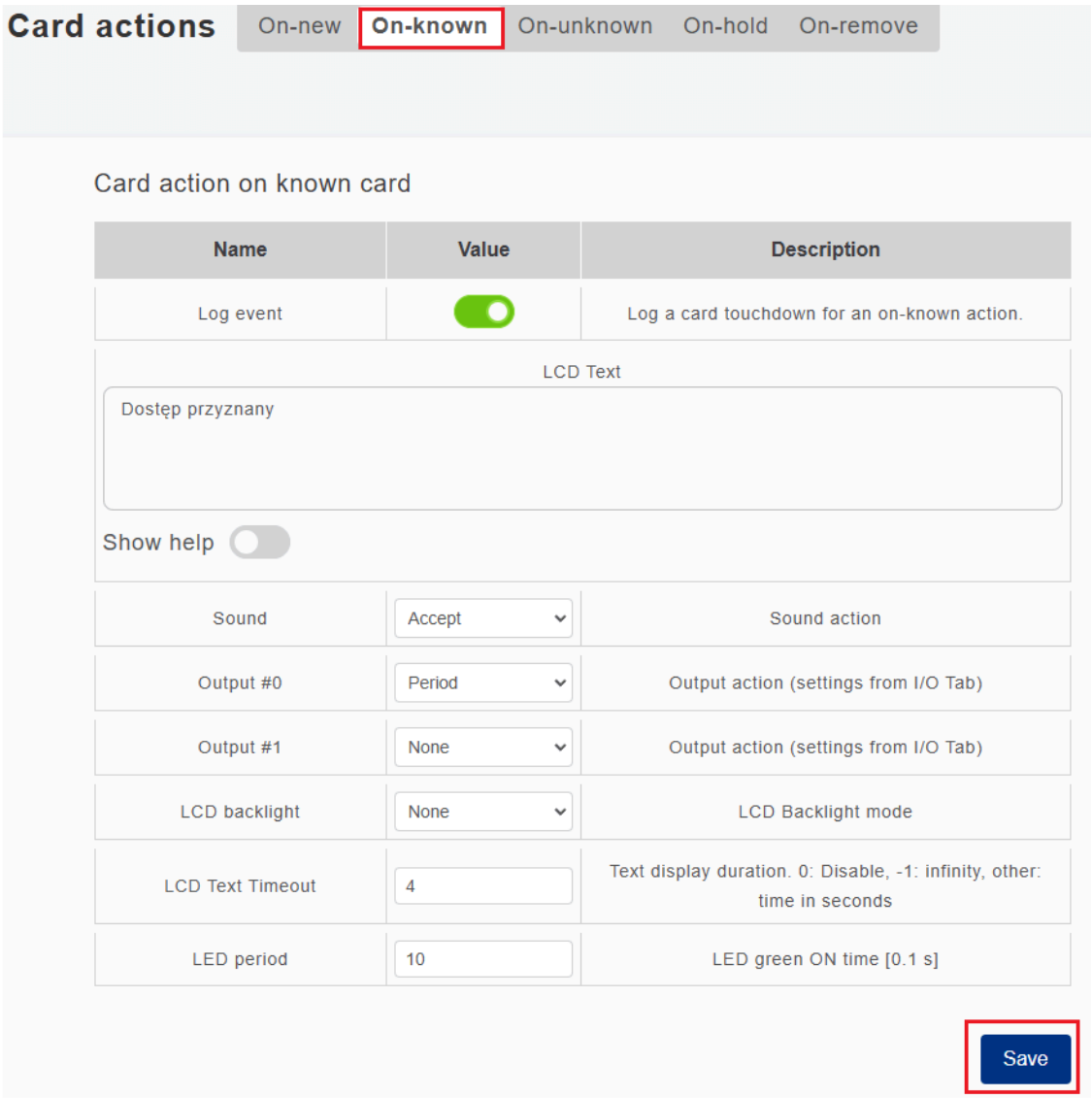
Name	Value	Description
Out #0 Mode	On period	Output #0 action
Out #0 Time	20	Time [0.1 s]
Out #0 Time Long	100	Time [0.1 s]
Out #0 Invert	None	Invert output

3. Defining card read action

- Go to the **Card actions** tab,
- In the **On-known** section (action on a known card) select:

- **Sound:** Accept (card accepted signal),
- **Output #0:** period (relay activated for a predefined period),
- **LCD text:** `Access granted`,
- **LCD Text Timeout** 5 seconds (text display time).

- **Save changes**



4. Testing the configuration

- Apply the previously added RFID card to the reader.
- The device should make a sound, display the `Access Granted` message and activate the relay for 2 seconds.
- Check if the relay drives the door opener correctly.

Result:
When an authorized user applies a card to the reader, the doors are unlocked for a predefined period of time. Access will be denied if a card is not present in the database.

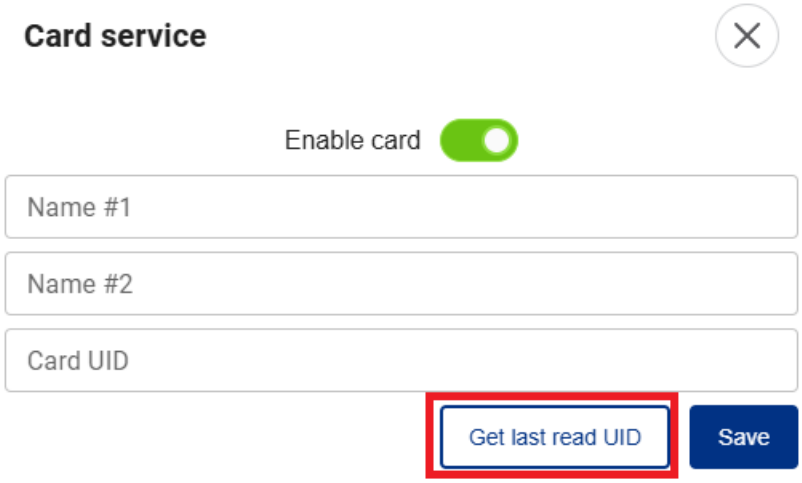
Example 2: Recording of work time

Objective: Using the BoxID LED HF for recording entering and exiting employees with the use of RFID cards.

Configuration:

1. Adding users and RFID cards

- Go to the **Card database** tab in the device interface,
- **Add a new RFID card to database** and assign a user name,
- Apply the RFID card to the reader - press **Get last read UID**,
- **Save changes**. Repeat for each user.



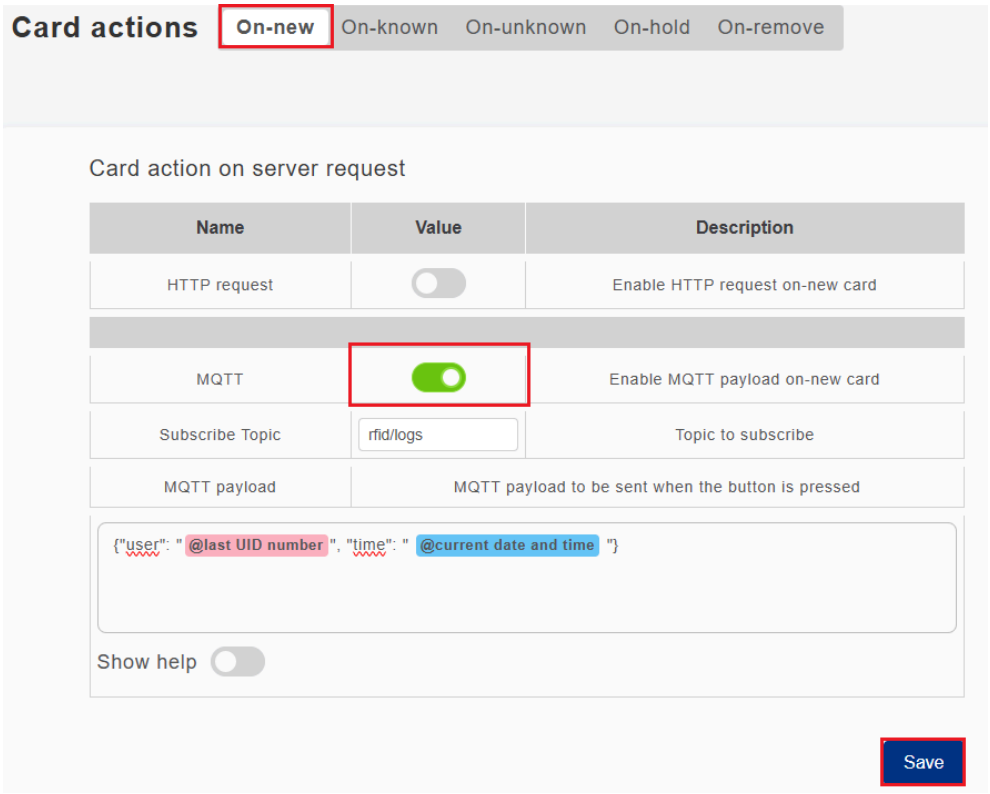
2. Event logging configuration

- Go to the **Card actions** tab,
- In the **On-known** section, set:
  - **Sound:** Accept (card accepted signal),
  - **LCD text:** `Hello, [name]!`,
  - **Log event: Enabled** (enables event logging in the system),
- In the **On unknown** section, set:
- **Sound:** Reject (card rejected signal),
- **LCD text:** `Card unknown!`,
- **Save changes**.

3. Sending data to the server

- Go to the **Services/MQTT** tab and enable event sending with MQTT,
- Configure the sever address (Broker) to which data about applied cards should be sent,
- In the **On-new** section, add information about the user to be sent:
- **MQTT topic:** `rfid/logs`,

- **MQTT payload:** `{"user": "@last UID number", "time": " @current date and time"}`.



4. Testing the configuration

- Apply the card to the reader,
- The screen should display the welcome message, and log the event in the system,
- Make sure the data has been sent correctly or saved in device logs.

Result:
Every RFID card use is logged and can be used to record working hours. Integration with a HR or ERP system allows for automatic computing of work time and generating a report.

Example 3: Automatic lighting control

Objective: Using the BoxID LED HF to control lighting with RFID card authorization.

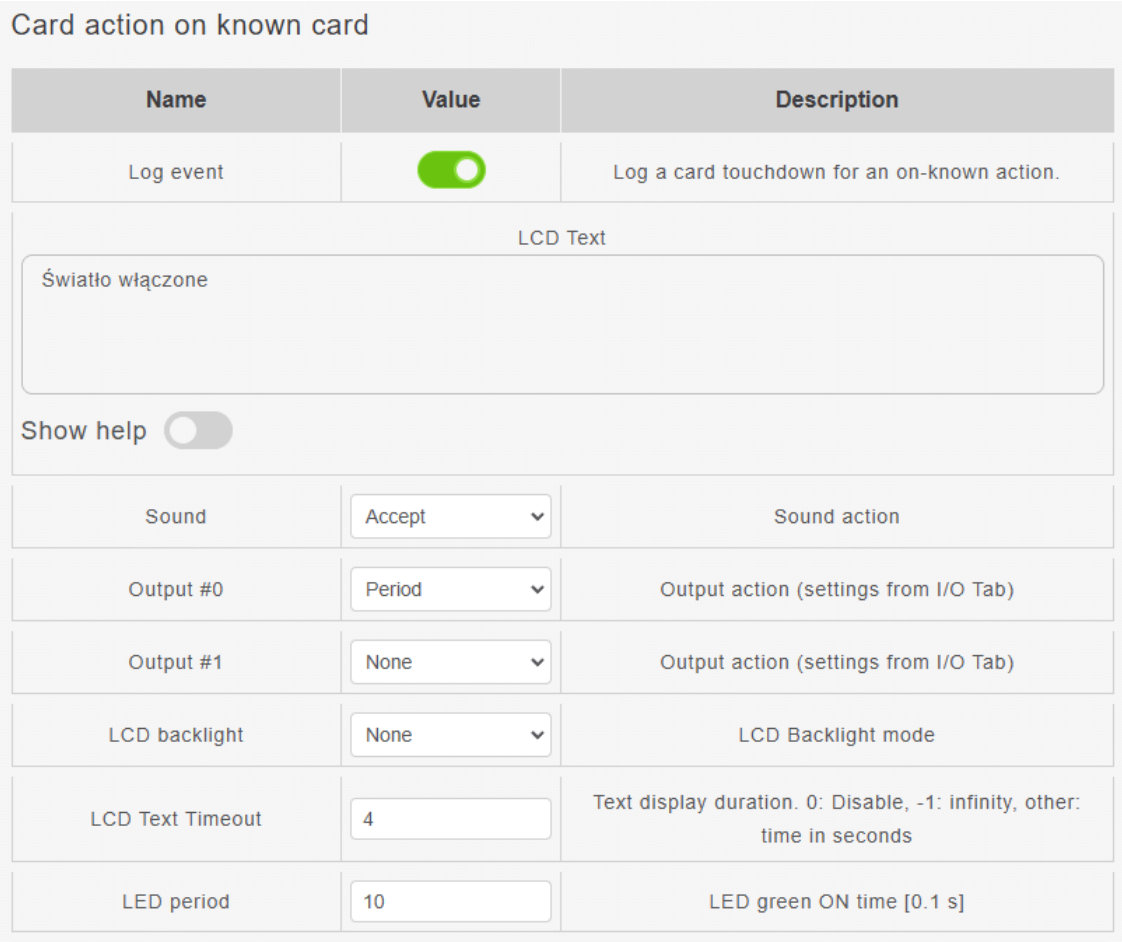
Configuration:

1. Connecting the device to the lighting system

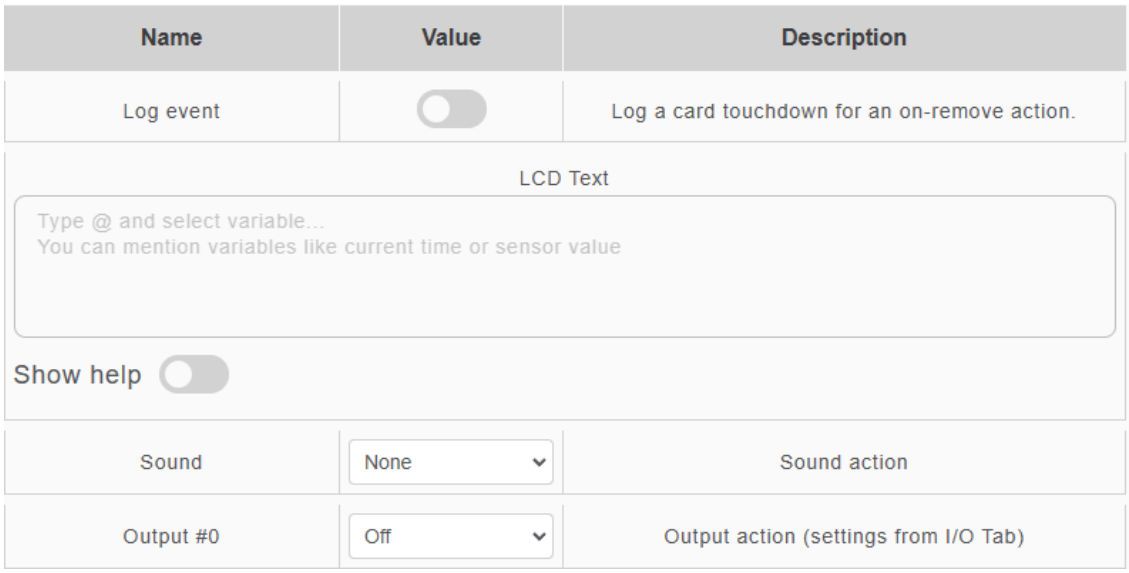
- Connect the relay controlling the lighting to **OUT1** or **OUT2** device output,
- Check if the relay turns the lighting on an off correctly.

2. Setting the RFID card read reaction

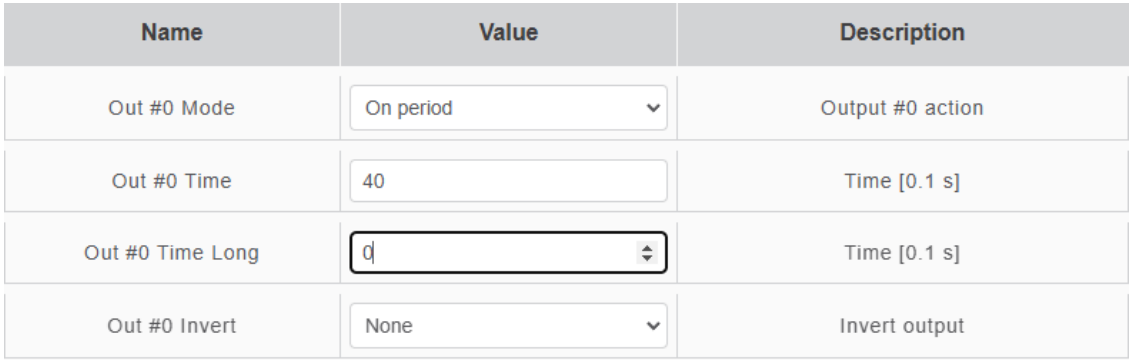
- Go to the **Card actions** tab,
- In the **On-known** section, set:
  - **Sound:** Accept (card accepted signal),
  - **LCD text:** `Lights ON`,
  - **Output #0:** `On Period` (relay activated for a predefined time),



- In the **On-remove** section, set:
  - **Sound:** None,
  - **LCD text:** `Lights OFF`,
  - **Output #0:** `Off` (relay deactivation after the card is removed).



- Go to the **I/O Settings** tab,
- In the **Channels** section, set:
  - **Out #0 Time** - eg. `3000` (5 minutes),
- **Save changes**.



3. Testing the configuration

- Apply the card to the reader,
- The lighting should be turned on for a predefined time,
- After the card is removed, the lighting should be turned off (If **On-remove** has been configured).

Result:
The system allows for automatic lighting control eg. in offices, warehouses, or staff rooms. The lighting is activated only when it is needed, which makes energy saving possible.

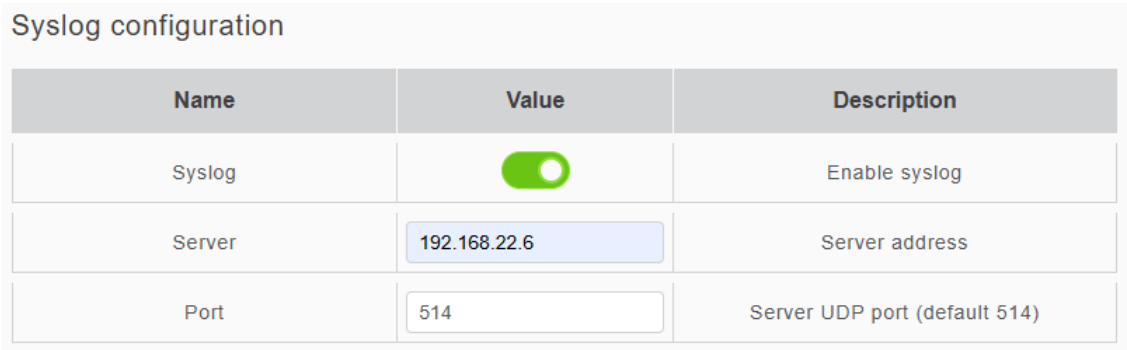
Example 4: Recording of access attempts by unauthorized cards and sending the events to the Syslog server

Objective:
Monitoring of access attempt by use of unauthorized RFID cards and automatic transfer of information about such events to the central Syslog server.

Configuration:

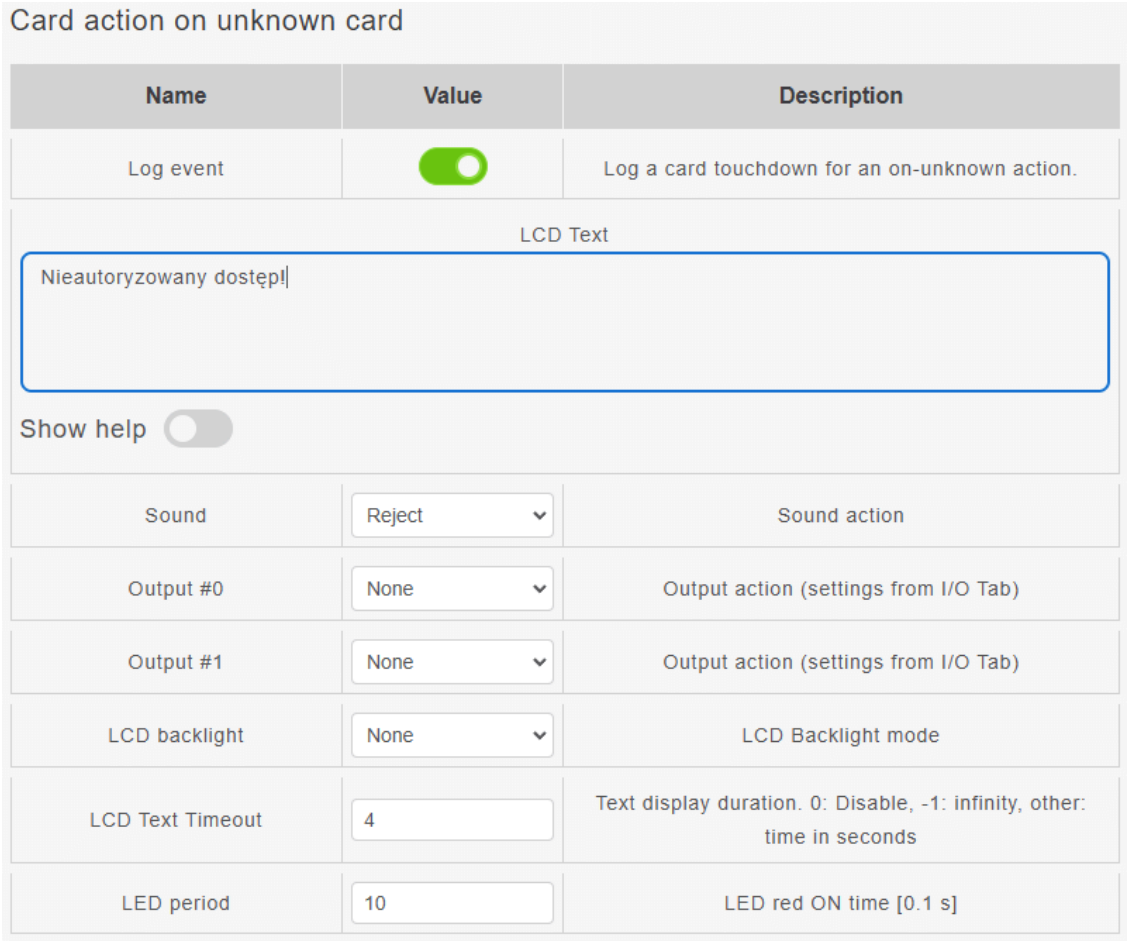
1. Enabling syslog

- Go to **Services/Syslog** tab in the device interface,
- In the **Syslog Settings** section:
  - **Enable:** Set `ON` to activate Syslog client,
  - **Server:** Enther the IP address or host name of the Syslog server,
  - **Port:** Set port number (default 514).
- **Save changes**.



2. Configuring the reaction for unauthorized cards

- Go to the **Card actions** tab,
- In the **On-unknown** section (action for unknown cards) set:
- **Sound:** Reject (card rejected signal),
- **LCD Text:** `Unauthorized access!` – message displayed on the screen,
- **Log event:** Make sure this option is enabled, for the event to be logged and sent to the Syslog server.
- **Save changes**.



3. Testing the configuration

- Use an RFID card that is not saved in the device database (unauthorized).
- The device should:
  - Make a card rejected signal,
  - Display the `Unauthorized access!` message,
  - Register the event locally and send it to the configured Syslog server.

Result:
The system automatically monitors and registers access attempts with unauthorized RFID cards. Events sent to the Syslog server allow for central monitoring and swift reaction in the case an unauthorized access is detected.

Example 5: Basic access control system using Lua script

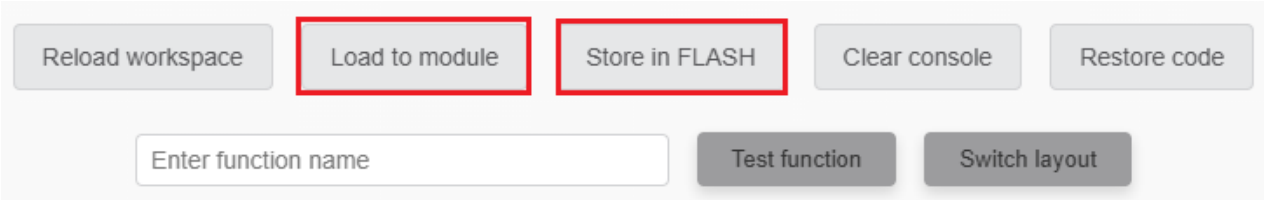
Description:
The following script demonstrates the basic access control logic using functions of the BoxID LED HF. The script executes the following actions:

- **rfid_detect()**
  Function enabled when the RFID subsystem detects a card and reads its UID. In this function, the UID read may be modified or simply recorded.
- **action_oncard()**
  Function recalled when the system recognizes a card. A check is conducted, whether the UID corresponds to an authorized card (eg. "12345678"). If yes, `action_known()` function is recalled. If not, `action_unknown` is recalled.
- **action_known()**
  If a card is authorized, the system grants access - displays the "**Access Granted**" message, activates the relay output (eg. opening the door), plays the card accepted signal, and sets a timeout after which the output is deactivated.
- **action_unknown()**
  If a card is unauthorized, the system rejects access - displays the "**Access Denied**" message, plays the card rejected signal and blocks the access attempt.
- **action_hold()**
  Enabled when a card remains near the reader for too long. In an extended version:
  - Displays the "**Card Hold Mode**" message,
  - Flashes the relay output (channel 0) – 3 cycles, 200 ms on, 200 ms off,
  - Logs the long card hold event.
- **action_remove()**
  Function recalled when the card is **removed** from the read area. Frequently used to clear the display and log the event.

Implementation instructions:

1. **Open the Lua Scripts tab:**
   Log in to the BoxID LED HF interface and go to **Lua Scripts** tab.
2. **Paste the script:**
   Copy the following code to the **script editor**.
3. **Save and upload:**
   Save the changes and upload the script to device using the **Load to module** button.
4. **Test the operation:**
   Apply a card to the reader:
   - If card UID is "12345678", the system grants access, activates the relay output, displays the **Access granted** message and plays the card accepted signal.

- For other UID's, the system denies access, displays the **Access Denied** message and logs the access attempt.
- Holding the card will recall the `action_hold()` function, removing it - `action_remove()`.
- Hold the card for a longer time to recall the `action_hold()` function. You will see the "**Card hold mode**" message, and the relay output will start to flash.
- Removing the card will recall the `action_remove()` function, which clears the display.



Warning

Remember that the Load to module button loads the script to RAM only. To permanently save changes after loading them to RAM, press the Store in FLASH button.

Lua Script:

```lua
-- Basic functions that are always executed.
-- Important! Even if you do not use the function, do not delete it; just leave it empty.

-- Function recalled when the RFID subsystem detects a card and reads its UID.
function rfid_detect()
    local uid = rfid.getUidAscii()  -- Read card UID in ASCII format.
    dev.console("RFID detected, UID: " .. uid)
    -- Optionally you can modify the UID, eg. convert it to the decimal format.
end

-- Function recalled when the system recognizes the card.
function action_oncard()
    local uid = rfid.getUidAscii()
    dev.console("Card presented, UID: " .. uid)
    -- Checking if the card is authorized (UID eg. "12345678")
    if uid == "12345678" then
        action_known()
    else
        action_unknown()
    end
end

-- Function recalled when the card is recognized as authorized.
function action_known()
    dev.console("Known card. Access granted for UID: " .. rfid.getUidAscii())
    dev.lcd_clear()
    dev.lcd_text("Access Granted", 1)
    dev.out_on(0)          -- Activate the relay output (eg. open the door)
    dev.buzz(0)            -- Play the card accepted signal (tone 0)
    dev.set_timeout(3000)  -- Set timeout to 3000 ms (3 seconds)
end

-- Function recalled when a card is recognized as unauthorized.
function action_unknown()
    dev.console("Unknown card. Access denied for UID: " .. rfid.getUidAscii())
    dev.lcd_clear()
    dev.lcd_text("Access Denied", 1)
    dev.buzz(1)           -- Play the card rejected signal (tone 1)
    dev.log("Unauthorized access attempt, UID: " .. rfid.getUidAscii())
end

-- Function recalled when the card remains in the read area for a longer time.
function action_hold()
    dev.console("Card held too long.")
    dev.lcd_clear()
    dev.lcd_text("Card Hold Mode", 1)
    -- Additional option: flashing the relay output (channel 0).
    dev.out_blink(0, 200, 200, 3)  -- 3 cycles: 200 ms on, 200 ms off
    dev.log("Card held for extended period, UID: " .. rfid.getUidAscii())
end

-- Function recalled, when a card is removed from the read area.
function action_remove()
    dev.console("Card removed.")
    dev.lcd_clear()
end
```

Example 6: Transfer of device data via HTTP

Objective: Transfering device data in the form of HTTP requests: ping, card application and removal information.

Configuration:

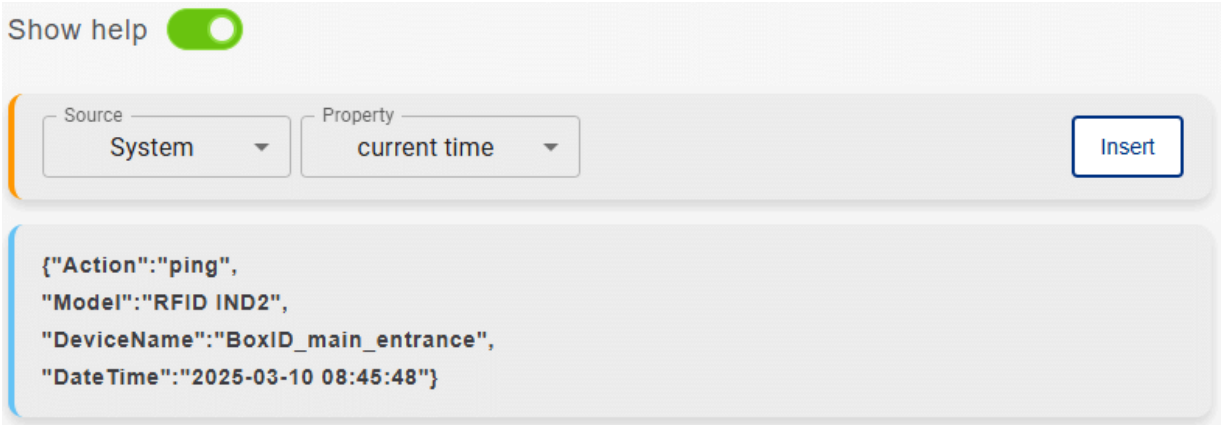
1. Setting HTTP parameters and adding the ping request syntax

- Go to the **Services / HTTPc** tab,
- Enable **HTTP Client**,
- Enter the server parameters: IP address into the **Server** field and **Port**,
- Set the selected **HTTP Method** and **Content type** (we will be using the **POST** method and **json** content type for the purpose of this demonstration),
- In the **Resource** field, enter the resource, to which the data will be sent,
- Depending on your needs, enter the user name and password to the corresponding fields,
- Set the time interval between consecutive ping requests (in seconds),
- Type the request syntax into the **HTTP ping request** field, eg.

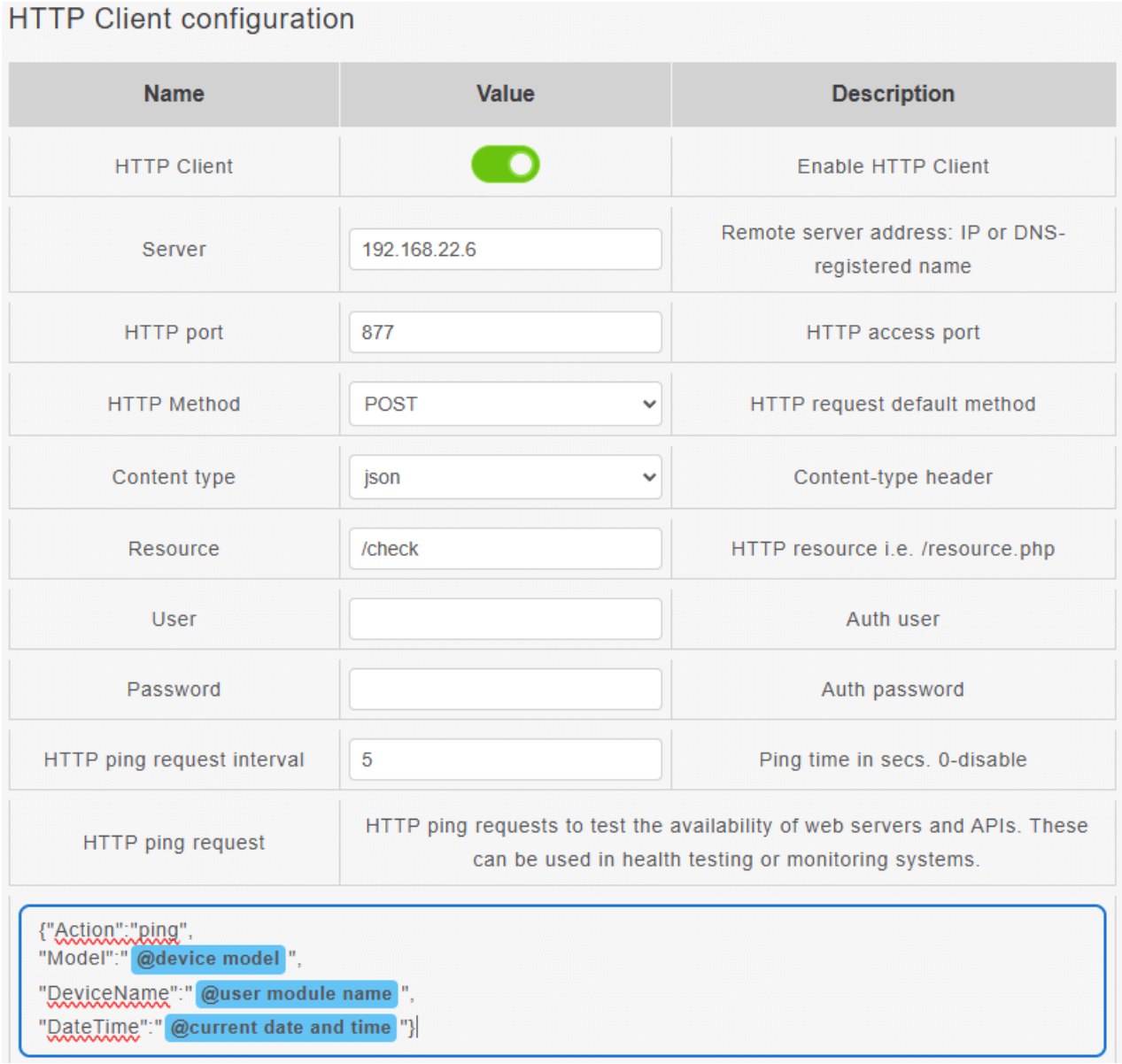
```
{"Action":"ping",
"Model":"%mod_model%",
"DeviceName":"%mod_name%",
"DateTime":"%timedate%"}
```

Tip

Syntax preview is possible after clicking the Show help **button:**



- Set the encryption parameters and upload certificates if needed,
- **Save the changes**.

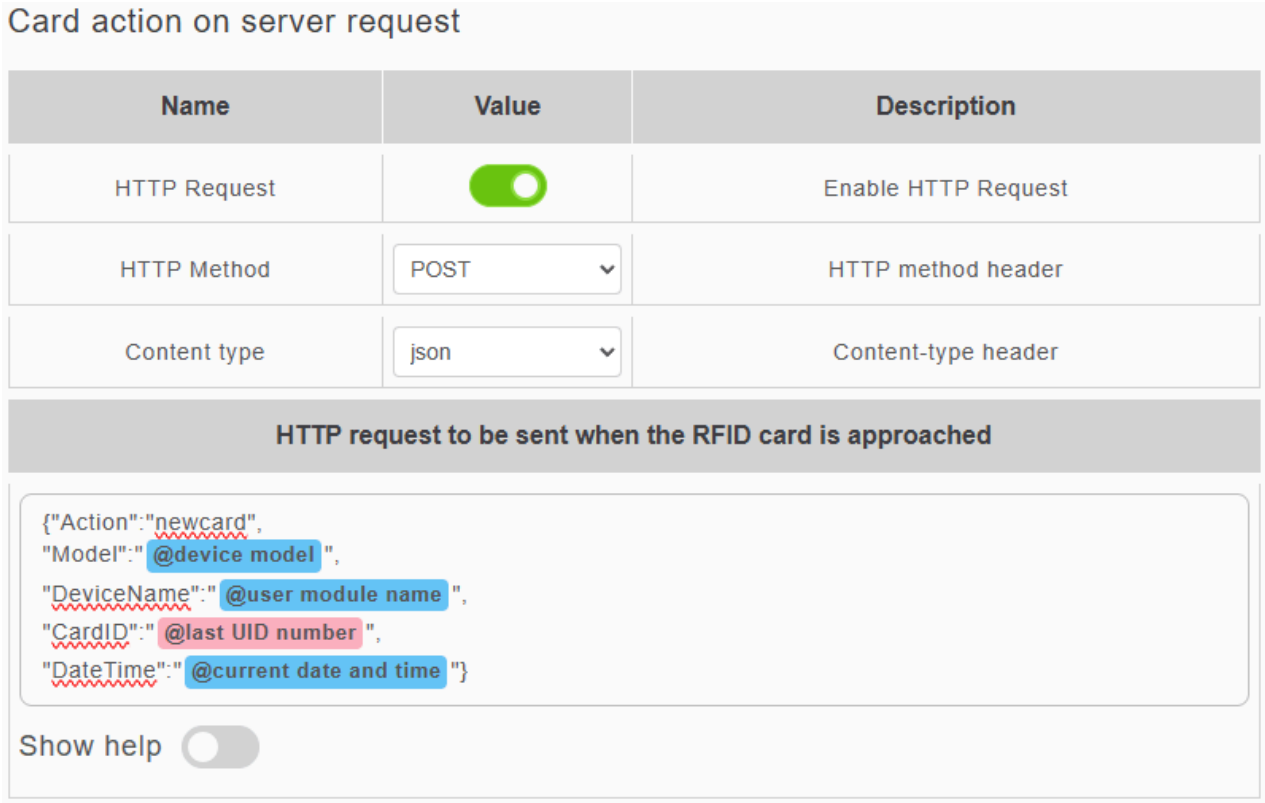


2. HTTP request actions setting

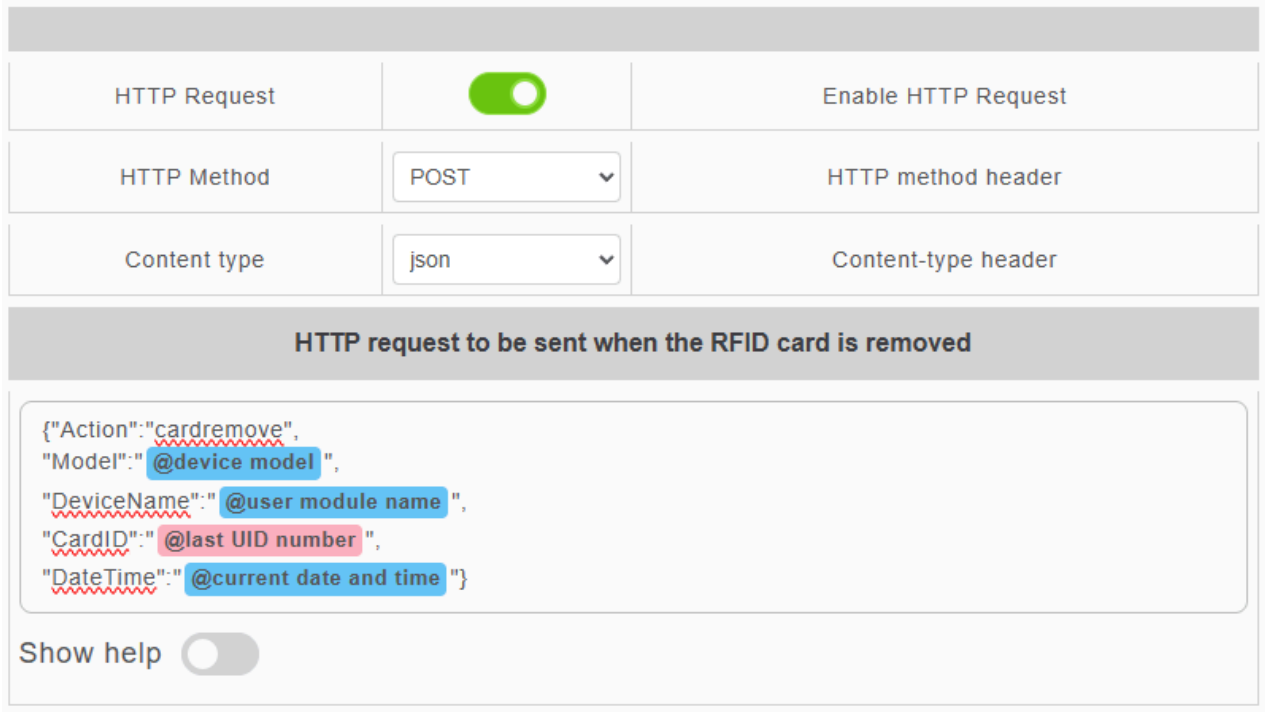
- Go to the **Card actions** tab and select the desired action (**On-new**, informing about card application in this case),
- Select **HTTP Request**,
- Choose the HTTP method,
- Select the data format,
- Enter the request syntax eg.:

```
{"Action":"newcard",
"Model":"%mod_model%",
"DeviceName":"%mod_name%",
"CardID":"%uid%",
"DateTime":"%timedate%"}
```

- **Save the changes**

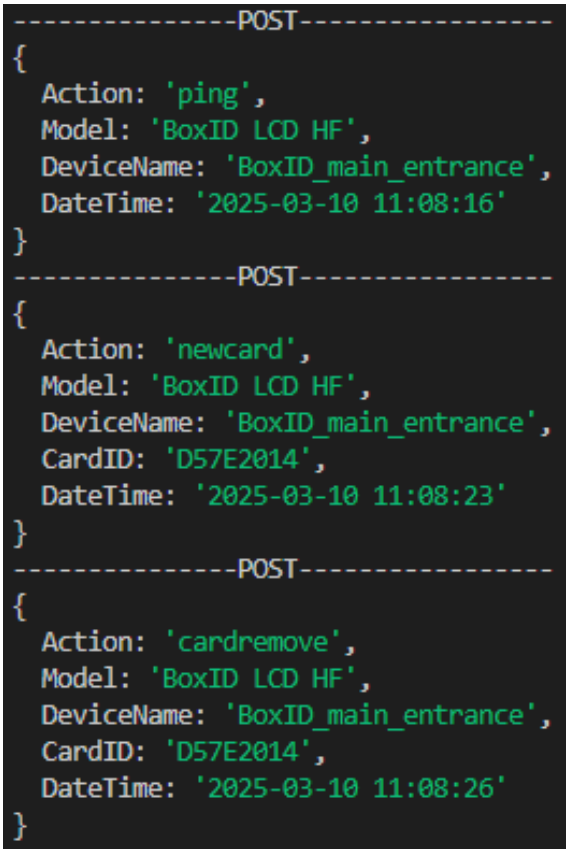


Likewise, configure the **On-hold** and **On-remove** actions if needed:



3. Test the configuration

- In the HTTP server's terminal, verify if the BoxID sends:
- Ping requests according to the interval set,
- Information about RFID card application to the read field,
- Information about card removal from the read field.



Result:

The device sends ping requests at specified intervals. Moreover, it informs the server about the fact that a card has been applied/removed to/from the read field, taking note of the event time and date.

Factory settings, backup

Restoring factory settings

To restore the device to factory settings:

- Power up the device.
- Short the **RESET** jumper for a time between 10 and 15 seconds.
- Remove the **RESET** jumper when the signal sounds.

> **Information**
>
> **Factory settings can be restored by using the built-in www server. The user should go to the Administration tab, then in Backup tab press the Reset to default button, which is located at the bottom of the page.**

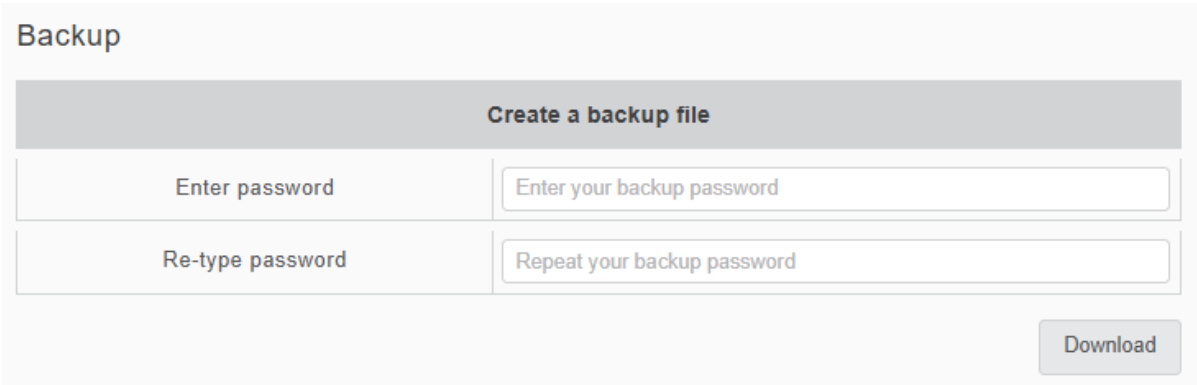
Once the above steps have been completed, the device will be set to the following parameters:

- **IP address:** 192.168.111.15
- **IP mask:** 255.255.255.0
- **User:** admin
- **Password:** admin

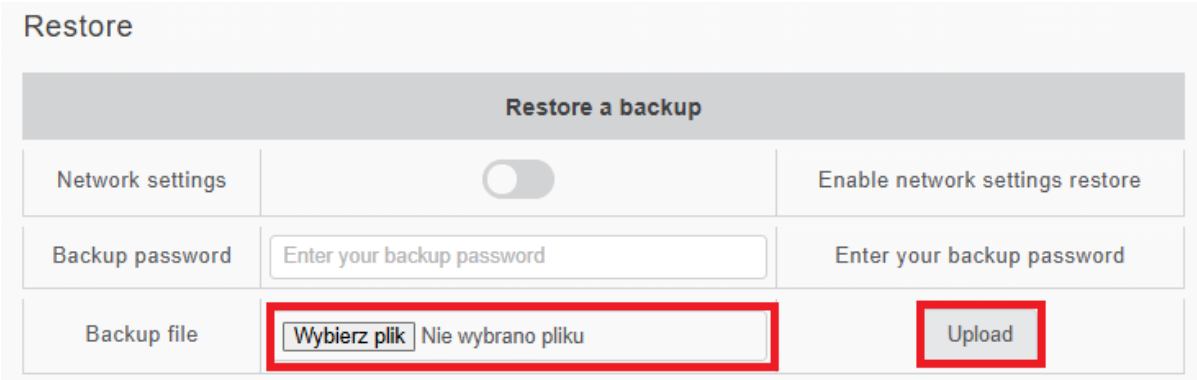
Backup / Restoring user settings

All configuration data can be saved to a file. The user can create/upload the backup file on the **Administration / Backup** page.

Backup
Creation of a configuration data save requires a password.



Uploading the backup
Uploading the backup configuration data requires a password and selecting the file from the computer using the **Choose file** and **Upload** buttons.



Emergency firmware restore

To restore firmware after a failure:

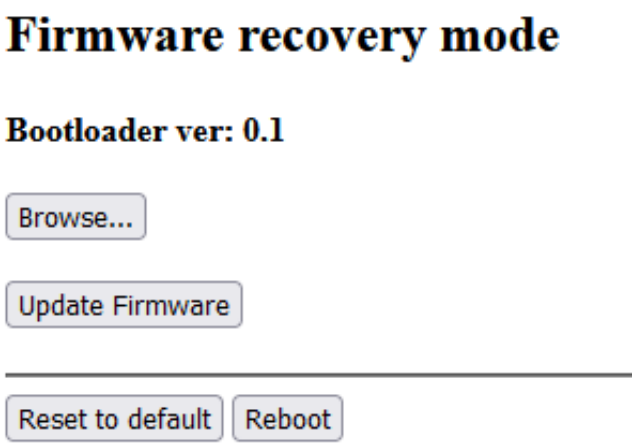
- Short the **RESET** jumper.
- Power up the device and connect it to LAN.

Once the above steps have been completed, the device will be set to the following parameters:

- **IP address**: 192.168.111.15
- **IP mask**: 255.255.255.0

> **Tip**
>
> **In order to access the 192.168.111.15 address, computer's IP address must be in the same subnet (example computer address: 192.168.111.1).**

Referring to the specified IP address will give access to the device's bootloader.



In this location it is possible to upload firmware, reset the device to factory settings and reboot.

Warranty and manufacturer's liability

> **Warning**
>
> **The manufacturer provides a two-year warranty for the device and post-warranty service for a period of 10 years from the date of introduction of the device to the market. The warranty covers all material and production defects.**

The manufacturer undertakes to respect the warranty agreement, if the following conditions are met:

- All repairs, changes, expansions and device calibrations are carried out by the manufacturer or an authorized service center,
- The power supply system meets the applicable standards,
- The device is operated in accordance with the suggestions presented in this manual,
- The device is operated in accordance with its intended purpose.

The manufacturer assumes no responsibility for consequences resulting from improper installation, improper use of the device, failure to comply with the instruction manual, and repairs made by unauthorized personnel.

> **Warning**
>
> **The device contains no user serviceable parts inside.**

Storage, operation and transport conditions

The device should be stored in enclosed rooms, where the atmosphere is free from vapours and corrosive substances:

- Environment temperature from -30°C to +60°C (-22°F - 140°F),
- Humidity from 25% to 90% (condensation unacceptable),
- Atmospheric pressure from 700 to 1060 hPa.

The device is intended to operate in the following conditions:

- Environment temperature from -10°C do +55°C (14°F - 131°F),
- Humidity from 30% to 75%,
- Atmospheric pressure from 700 to 1060 hPa.

Recommended transport conditions:

- Environment temperature from -40°C do +85°C (-40°F - 185°F),
- Humidity from 5% to 95%,
- Atmospheric pressure from 700 to 1060 hPa.

Installation and device operation:

- The module should be operated in accordance with recommendations provided later in this manual.

Disposal and decommissioning

In an event the device needs to be decommissioned (eg. after its intended life period is surpassed), it is recommended to contact the manufacturer or his representative, who are responsible to respond appropriately, i.e., to collect the device from the user. The user can alternatively contact companies specializing in electronic device or computer equipment disposal and/or decommissioning. Under no condition should the device be placed with other waste.