HTTP API manual for 2N IP intercoms

Configuration Manual

Firmware: 2.33
Version: 2.33

www.2n.cz

The 2N TELEKOMUNIKACE a.s. is a Czech manufacturer and supplier of telecommunications equipment.

The product family developed by 2N TELEKOMUNIKACE a.s. includes GSM gateways, private branch exchanges (PBX), and door and lift communicators. 2N TELEKOMUNIKACE a.s. has been ranked among the Czech top companies for years and represented a symbol of stability and prosperity on the telecommunications market for almost two decades. At present, we export our products into over 120 countries worldwide and have exclusive distributors on all continents.

2N TELEKOMUNIKACE a.s. administers the FAQ database to help you quickly find information and to answer your questions about 2N products and services. On www.faq.2n.cz you can find information regarding products adjustment and instructions for optimum use and procedures „What to do if...".

2N TELEKOMUNIKACE a.s. hereby declares that the 2N product complies with all basic requirements and other relevant provisions of the 1999/5/EC directive. For the full wording of the Declaration of Conformity see the CD-ROM (if enclosed) or our website at www.2n.cz.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

The 2N TELEKOMUNIKACE a.s. is the holder of the ISO 9001:2009 certificate. All development, production and distribution processes of the company are managed by this standard and guarantee a high quality, technical level and professional aspect of all our products.

Content:

- 1. Introduction
  - 1.1 HTTP API Release Notes
- 2. HTTP API Description
  - 2.1 HTTP Methods
  - 2.2 Request Parameters
  - 2.3 Replies to Requests
- 3. HTTP API Services Security
- 4. User Accounts
- 5. Overview of HTTP API Functions
  - 5.1 api system
    - 5.1.1 api system info
    - 5.1.2 api system status
    - 5.1.3 api system restart
    - 5.1.4 api system caps
  - 5.2 api firmware
    - 5.2.1 api firmware
    - 5.2.2 api firmware apply
    - 5.2.3 api firmware reject
  - 5.3 api config
    - 5.3.1 api config
    - 5.3.2 api config factoryreset
    - 5.3.3 api config holidays
  - 5.4 api switch
    - 5.4.1 api switch caps
    - 5.4.2 api switch status
    - 5.4.3 api switch ctrl

- 5.5 api io
  - 5.5.1 api io caps
  - 5.5.2 api io status
  - 5.5.3 api io ctrl
- 5.6 api phone
  - 5.6.1 api phone staus
- 5.7 api call
  - 5.7.1 api call status
  - 5.7.2 api call dial
  - 5.7.3 api call answer
  - 5.7.4 api call hangup
- 5.8 api camera
  - 5.8.1 api camera caps
  - 5.8.2 api camera snapshot
- 5.9 api display
  - 5.9.1 api display caps
  - 5.9.2 api display image
    - 5.9.1 api display image examples
- 5.10 api log
  - 5.10.1 api log caps
  - 5.10.2 api log subscribe
  - 5.10.3 api log unsubscribe
  - 5.10.4 api log pull
- 5.11 api audio
  - 5.11.1 api audio test
- 5.12 api email
  - 5.12.1 api email send
- 5.13 api pcap
  - 5.13.1 api pcap
  - 5.13.2 api pcap restart
  - 5.13.3 api pcap stop
  - 5.13.4 api pcap live
  - 5.13.5 api pcap live stop
  - 5.13.6 api pcap live stats

- 5.14 api dir
  - 5.14.1 api dir template
  - 5.14.2 api dir create
  - 5.14.3 api dir update
  - 5.14.4 api dir delete
  - 5.14.5 api dir get
  - 5.14.6 api dir query
- 5.15 api mobilekey
  - 5.15.1 api mobilekey config
- 5.16 api lpr
  - 5.16.1 api lpr licenseplate
  - 5.16.2 api lpr image
- 5.17 api accesspoint blocking
  - 5.17.1 api accesspoint blocking ctrl
  - 5.17.2 api accesspoint blocking status

1. Introduction

HTTP API is an application interface designed for control of selected 2N IP intercoms functions via the HTTP. It enables 2N IP intercoms to be integrated easily with third party products, such as home automation, security and monitoring systems, etc.

HTTP API provides the following services:

- System API – provides intercom configuration changes, status info and upgrade.
- Switch API – provides switch status control and monitoring, e.g. door lock opening, etc.
- I/O API – provides intercom logic input/output control and monitoring.
- Audio API – provides audio playback control and microphone monitoring.
- Camera API – provides camera image control and monitoring.
- Display API – provides display control and user information display.
- E-mail API – provides sending of user e-mails.
- Phone/Call API – provides incoming/outgoing call control and monitoring.
- Logging API – provides reading of event records.

Set the transport protocol (HTTP or HTTPS) and way of authentication (None, Basic or Digest) for each function. Create up to five user accounts (with own username and password) in the HTTP API configuration for detailed access control of services and functions.

Use the configuration web interface on the Services / HTTP API tab to configure your HTTP API. Enable and configure all the available services and set the user account parameters.

Refer to http(s)://ip_intercom_address/apitest.html for a special tool integrated in the intercom HTTP server for HTTP API demonstration and testing.

Caution

Warning

In order to ensure the full functionality and guaranteed performance, we strongly recommend that the topicality of the product / device version in use be verified as early as in the installation process. The customer hereby acknowledges that the product / device can achieve the guaranteed performance and full functionality pursuant to the manufacturer's instructions only if the latest product / device version is used after having been tested for full interoperability and not having been determined by the manufacturer as incompatible with certain versions of other products, and only in conformity with the manufacturer's instructions, guidelines or recommendations and in conjunction with suitable products and devices of other suppliers. The latest versions are available at https://www.2n.cz/cs_CZ/ or can be updated via the configuration interface if the devices are adequately technically equipped. Should the customer use a product / device version other than the latest one or a version determined by the manufacturer as incompatible with certain versions of other products, or should the customer use the product / device in contradiction to the manufacturer's instructions, guidelines or recommendations or in conjunction with unsuitable products / devices of other suppliers, the customer is aware of and agrees with all functionality limitations of such a product / device if any as well as with all consequences incurred as a result thereof. Using a product / device version other than the latest one or a version determined by the manufacturer as incompatible with certain versions of other products, or using the product / device in contradiction to the manufacturer's instructions, guidelines or recommendations or in conjunction with unsuitable products / devices of other suppliers, the customer agrees that the 2N TELEKOMUNIKACE a.s. company shall not be held liable for any functionality limitation of such a product or any damage, loss or injury related to this potential functionality limitation.

1.1 HTTP API Release Notes

1.1 HTTP API Release Notes

Version	Changes
2.33	• Addition of the **/api/pcap/live**, **api/pcap/live/stop** and **api/pcap/stats** functions for incoming / outgoing packet capture control.
2.32	• Addition of the **/api/lpr/licenseplate** function for access control based on license plate recognition. • Addition of the **api/lpr/image** function for retrieving images received from license plate recognition.
2.31	• Addition of the **/api/mobilekey/config** for reading and writing location IDs and encryption keys for Bluetooth Authentication.
2.30	• Removal of the **apbBroken** parameter from the **AccessTaken** event. • Addition of the **apbBroken** parameter to the **UserAuthenticated** event.
2.29	• Addition of the new function for **api system caps**. • New event **CapabilitiesChanged**.
2.28	• Unchanged.
2.27	• New events: **LiftStatusChanged, LiftConfigChanged, LiftFloorEnabled.** • Addition of the new functions for **api holidays, api config holidays, api dir template, api dir create, api dir update, api dir delete, api dir get, api dir query**.
2.26	• New events: **DtmfEntered**, **AccessTaken**, **ApLockStateChanged**, **RexActivated**.
2.25	• Unchanged.
2.24	• Change of user addition to the directory due to deletion of positions.

Version	Changes
2.23	• Addition of the **switchDisabled** parametr to **UserRejected** event.
2.22	• New events: **CardHeld**, **PairingStateChanged**, **SwitchesBlocked**, **FingerEntered**, **MobKeyEntered**, **DoorStateChanged**, **UserRejected**, **DisplayTouched**.
2.21	• The **api/display/image** (**display**, **blob-image**, **blob-video**, **duration**, **repeat** parameters) function extended for **2N® IP Verso** • New events: **UserAuthenticated**, **SilentAlarm**, **AccessLimited** • Addition of the **timeSpan** parameter to the **/api/email/send** function
2.15	• New events: **TamperSwitchActivated**, **UnauthorizedDoorOpen**, **DoorOpenTooLong** and **LoginBlocked** • Addition of the **tzShift** event that gives the difference between the local time and Coordinated Universal Time (UTC) • The **email/send** function extended with a resolution setting option for the images to be sent
2.14	• Addition of the **api/pcap**, **api/pcap/restart** and **api/pcap/stop** functions for network traffic download and control • Addition of the **audio/test** function for automatic audio test launch • Addition of the **email/send** function • Addition of the **response** parameter to the **/api/io/ctrl** and **/api/switch/ctrl** functions • The **/call/hangup** function extended with a **reason** parameter specifying the call end reason • New events: **MotionDetected, NoiseDetected** and **SwitchStateChanged** • The **CallStateChanged** event extended with a **reason** parameter specifying the call end reason
2.13	• First document version

2. HTTP API Description

All **HTTP API** commands are sent via **HTTP/HTTPS** to the intercom address with absolute path completed with the **/api** prefix. Which protocol you choose depends on the current intercom settings in the **Services / HTTP API** section. The **HTTP API** functions are assigned to services with defined security levels including the **TLS** connection request (i.e. **HTTPS**).

Example: *Switch 1 activation* **http://10.0.23.193/api/switch/ctrl?switch=1&action=on**

The absolute path includes the function group name (system, firmware, config, switch, etc.) and the function name (caps, status, ctrl, etc.). To be accepted by the intercom, a request has to include the method and absolute path specification followed by the Host header.

Example:

```
GET /api/system/info HTTP/1.1
Host: 10.0.23.193
Intercom HTTP Server reply:
HTTP/1.1 200 OK
Server: HIP2.10.0.19.2
Content-Type: application/json
Content-Length: 253
{
  "success" : true,
  "result" : {
    "variant" : "2N Helios IP Vario",
    "serialNumber" : "08-1860-0035",
    "hwVersion" : "535v1",
    "swVersion" : "2.10.0.19.2",
    "buildType" : "beta",
    "deviceName" : "2N Helios IP Vario"
  }
}
```

This chapter also includes:

- 2.1 HTTP Methods
- 2.2 Request Parameters
- 2.3 Replies to Requests

2.1 HTTP Methods

2N IP intercom applies the following four HTTP methods:

- GET – requests intercom content download or general command execution
- POST – requests intercom content download or general command execution
- PUT – requests intercom content upload
- DELETE – requests intercom content removal

The GET and POST methods are equivalent from the viewpoint of HTTP API but use different parameter transfers (refer to the next subsection). The PUT and DELETE methods are used for handling of such extensive objects as configuration, firmware, images and sound files.

2.2 Request Parameters

Practically all the HTTP API functions can have parameters. The parameters (switch, action, width, height, blob-image, etc.) are included in the description of the selected HTTP API function. The parameters can be transferred in three ways or their combinations:

1. in the request path (uri query, GET, POST, PUT and DELETE methods);
2. in the message content (application/x-www-form-urlencoded, POST and PUT methods);
3. in the message content (multipart/form-data, POST and PUT methods) – RFC-1867.

If the transfer methods are combined, a parameter may occur more times in the request. In that case, the last incidence is preferred.

There are two types of the HTTP API parameters:

1. Simple value parameters (switch, action, etc.) can be transferred using any of the above listed methods and do not contain the blob- prefix.
2. Large data parameters (configuration, firmware, images, etc.) always start with blob- and can only be transferred via the last-named method (multipart/form-data).

2.3 Replies to Requests

Replies to requests are mostly in the JSON format. Binary data download (user sounds, images, etc.) and intercom configuration requests are in XML. The Content-Type header specifies the response format. Three basic reply types are defined for JSON.

Positive Reply without Parameters

This reply is sent in case a request has been executed successfully for functions that do not return any parameters. This reply is always combined with the HTTP status code 200 OK.

```
{
  "success" : true,
}
```

Positive Reply with Parameters

This reply is sent in case a request has been executed successfully for functions that return supplementary parameters. The result item includes other reply parameters related to the function. This reply is always combined with the HTTP status code 200 OK.

```
{
  "success" : true,
  "result" : {
    ...
  }
}
```

Negative Reply at Request Error

This reply is sent in case an error occurs during request processing. The reply specifies the error code (**code**), text description (**description**) and error details if necessary (**param**). The reply can be combined with the HTTP status code 200 OK or 401 Authorisation Required.

```
{
  "success" : false,
  "error" : {
    "code" : 12,
    "param" : "port",
    "description" : "invalid parameter value"
  }
}
```

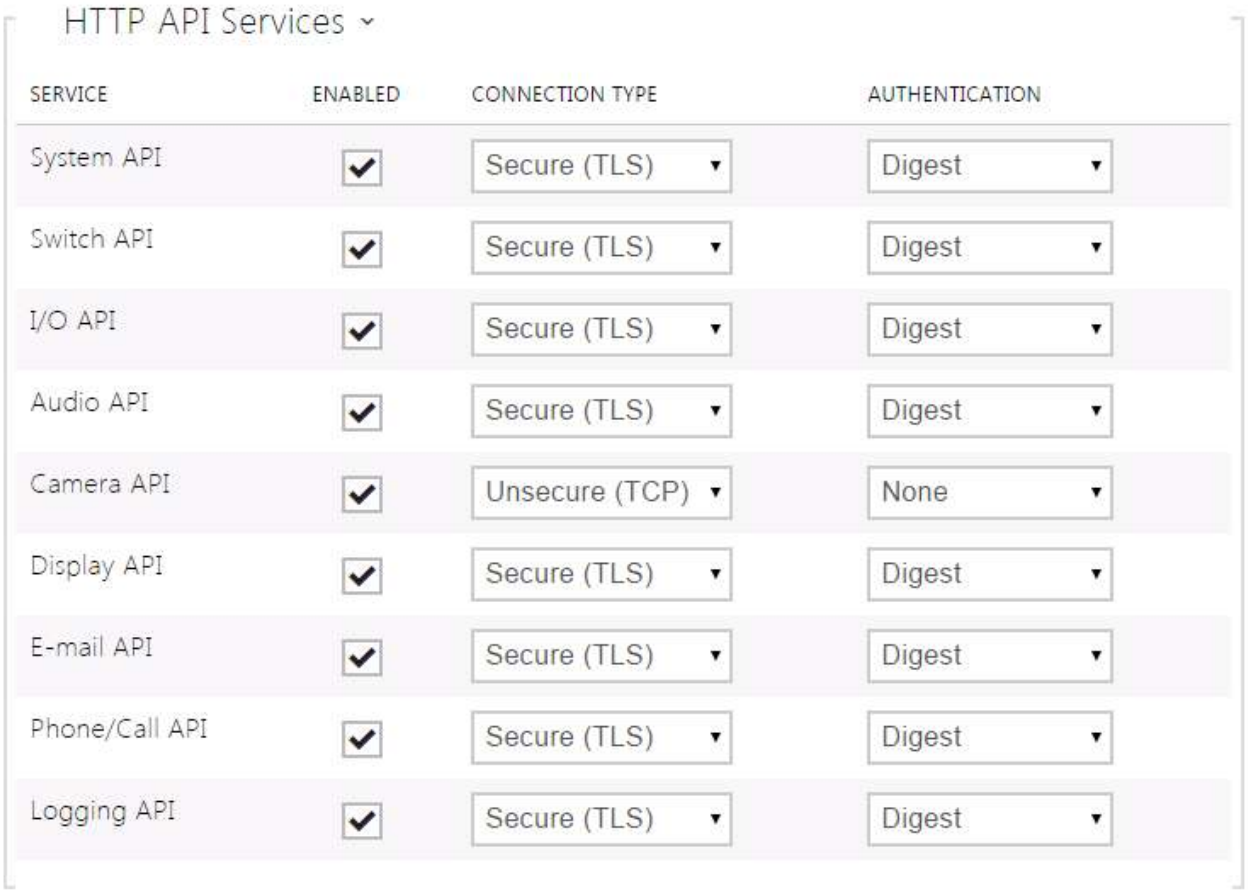
The table below includes a list of available error codes.

Code	Description	
1	function is not supported	The requested function is unavailable in this model.
2	invalid request path	The absolute path specified in the HTTP request does not match any of the HTTP API functions.
3	invalid request method	The HTTP method used is invalid for the selected function.
4	function is disabled	The function (service) is disabled. Enable the function on the Services / HTTP API configuration interface page.
5	function is licensed	The function (service) is subject to licence and available with a licence key only.
7	invalid connection type	HTTPS connection is required.

Code	Description	
8	invalid authentication method	The authentication method used is invalid for the selected service. This error happens when the Digest method is only enabled for the service but the client tries to authenticate via the Basic method.
9	authorisation required	User authorisation is required for the service access. This error is sent together with the HTTP status code Authorisation Required.
10	insufficient user privileges	The user to be authenticated has insufficient privileges for the function.
11	missing mandatory parameter	The request lacks a mandatory parameter. Refer to param for the parameter name.
12	invalid parameter value	A parameter value is invalid. Refer to param for the parameter name.
13	parameter data too big	The parameter data exceed the acceptable limit. Refer to param for the parameter name.
14	unspecified processing error	An unspecified error occurred during request processing.
15	no data available	The required data are not available on the server.
17	parameter shouldn't be present	Parameter collision (it is impossible to write a specified parameter combination).
18	request is rejected	The request cannot be processed now and was rejected by the device.
19	file version is lower than minimum	The submitted file version is lower than required.

3. HTTP API Services Security

Set the security level for each HTTP API service via the 2N IP intercom configuration web interface on the Services / HTTP API tab: disable/enable a service and select the required communication protocol and user authentication method.



Set the required transport protocol for each service separately:

- HTTP – send requests via HTTP or HTTPS. Both the protocols are enabled and the security level is defined by the protocol used.
- HTTPS – send requests via HTTPS. Any requests sent via the unsecured HTTP are rejected by the intercom. HTTPS secures that no unauthorised person may read the contents of sent/received messages.

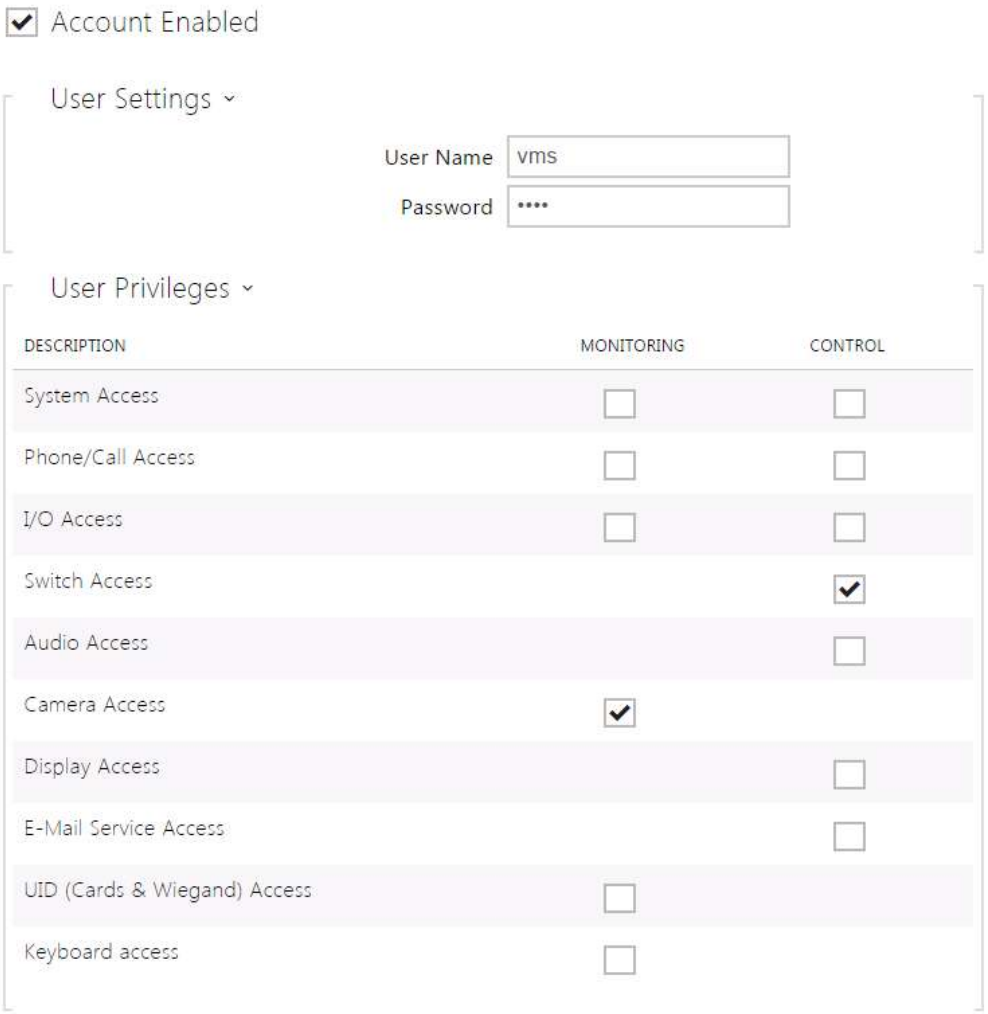
Set authentication methods for the requests to be sent to the intercom for each service. If the required authentication is not executed, the request will be rejected. Requests are authenticated via a standard authentication protocol described in RFC-2617. The following three authentication methods are available:

- None – no authentication is required. In this case, this service is completely unsecure in the LAN.
- Basic – Basic authentication is required according to RFC-2617. In this case, the service is protected with a password transmitted in an open format. Thus, we recommend you to combine this option with HTTPS where possible.
- Digest – Digest authentication is required according to RFC-2617. This is the default and most secure option of the three above listed methods.

We recommend you to use the HTTPS + Digest combination for all the services to achieve the highest security and avoid misuse. If the other party does not support this combination, the selected service can be granted a dispensation and assigned a lower security level.

4. User Accounts

With 2N IP intercom you can administer up to five user accounts for access to the HTTP API services. The user account contains the user's name, password and HTTP API access privileges.



Use the table above to control the user account privileges to the HTTP API services.

5. Overview of HTTP API Functions

The table below provides a list of all available HTTP API functions including:

- the HTTP request absolute path;
- the supported HTTP methods;
- the service in which the function is included;
- the required user privileges (if authentication is used);
- the required licence (Enhanced Integration licence key).

Absolute path	Method	Service	Privileges	Licence
/api/system/info	GET/POST	System	System Control	No
/api/system/status	GET/POST	System	System Control	Yes
/api/system/restart	GET/POST	System	System Control	Yes
/api/system/caps	POST	System	System Control	Yes
/api/firmware	PUT	System	System Control	Yes
/api/firmware/apply	GET/POST	System	System Control	Yes
/api/config	GET/POST/PUT	System	System Control	Yes
/api/config/factoryreset	GET/POST	System	System Control	Yes
/api/switch/caps	GET/POST	Switch	Switch Monitoring	Yes
/api/switch/status	GET/POST	Switch	Switch Monitoring	Yes

Absolute path	Method	Service	Privileges	Licence
/api/switch/ctrl	GET/POST	Switch	Switch Control	Yes
/api/io/caps	GET/POST	I/O	I/O Monitoring	Yes
/api/io/status	GET/POST	I/O	I/O Monitoring	Yes
/api/io/ctrl	GET/POST	I/O	I/O Control	Yes
/api/phone/status	GET/POST	Phone/Call	Call Monitoring	Yes
/api/call/status	GET/POST	Phone/Call	Call Monitoring	Yes
/api/call/dial	GET/POST	Phone/Call	Call Control	Yes
/api/call/answer	GET/POST	Phone/Call	Call Control	Yes
/api/call/hangup	GET/POST	Phone/Call	Call Control	Yes
/api/camera/caps	GET/POST	Camera	Camera Monitoring	No
/api/camera/snapshot	GET/POST	Camera	Camera Monitoring	No
/api/display/caps	GET/POST	Display	Display Control	Yes
/api/display/image	PUT/DELETE	Display	Display Control	Yes
/api/log/caps	GET/POST	Logging	*	No
/api/log/subscribe	GET/POST	Logging	*	No
/api/log/unsubscribe	GET/POST	Logging	*	No
/api/log/pull	GET/POST	Logging	*	No
/api/audio/test	GET/POST	Audio	Audio Control	Yes
/api/email/send	GET/POST	E-mail	E-mail Control	Yes

Absolute path	Method	Service	Privileges	Licence
/pcap	GET/POST	System	System Control	Yes
/pcap/restart	GET/POST	System	System Control	Yes
/pcap/stop	GET/POST	System	System Control	Yes
/api/holidays	GET/PUT	System	System Control	Yes
/api/config/holidays	GET/PUT	System	System Control	Yes
/api/dir/template	GET	System	System Control	Yes
/api/dir/create	PUT	System	System Control	Yes
/api/dir/update	PUT	System	System Control	Yes
/api/dir/delete	PUT	System	System Control	Yes
/api/dir/get	POST	System	System Control	Yes
/api/dir/query	POST	System	System Control	Yes
/api/mobilekey/config	GET/PUT	System	System Control	No
/api/lpr/licenseplate	POST	Camera	License Plate Recognition	No
/api/lpr/image	GET	System	System Control	No

This chapter also includes:

- 5.1 api system
  - 5.1.1 api system info
  - 5.1.2 api system status
  - 5.1.3 api system restart
  - 5.1.4 api system caps

- 5.2 api firmware
  - 5.2.1 api firmware
  - 5.2.2 api firmware apply
  - 5.2.3 api firmware reject
- 5.3 api config
  - 5.3.1 api config
  - 5.3.2 api config factoryreset
  - 5.3.3 api config holidays
- 5.4 api switch
  - 5.4.1 api switch caps
  - 5.4.2 api switch status
  - 5.4.3 api switch ctrl
- 5.5 api io
  - 5.5.1 api io caps
  - 5.5.2 api io status
  - 5.5.3 api io ctrl
- 5.6 api phone
  - 5.6.1 api phone staus
- 5.7 api call
  - 5.7.1 api call status
  - 5.7.2 api call dial
  - 5.7.3 api call answer
  - 5.7.4 api call hangup
- 5.8 api camera
  - 5.8.1 api camera caps
  - 5.8.2 api camera snapshot
- 5.9 api display
  - 5.9.1 api display caps
  - 5.9.2 api display image
    - 5.9.1 api display image examples
- 5.10 api log
  - 5.10.1 api log caps
  - 5.10.2 api log subscribe
  - 5.10.3 api log unsubscribe
  - 5.10.4 api log pull
- 5.11 api audio
  - 5.11.1 api audio test

- 5.12 api email
  - 5.12.1 api email send
- 5.13 api pcap
  - 5.13.1 api pcap
  - 5.13.2 api pcap restart
  - 5.13.3 api pcap stop
  - 5.13.4 api pcap live
  - 5.13.5 api pcap live stop
  - 5.13.6 api pcap live stats
- 5.14 api dir
  - 5.14.1 api dir template
  - 5.14.2 api dir create
  - 5.14.3 api dir update
  - 5.14.4 api dir delete
  - 5.14.5 api dir get
  - 5.14.6 api dir query
- 5.15 api mobilekey
  - 5.15.1 api mobilekey config
- 5.16 api lpr
  - 5.16.1 api lpr licenseplate
  - 5.16.2 api lpr image
- 5.17 api accesspoint blocking
  - 5.17.1 api accesspoint blocking ctrl
  - 5.17.2 api accesspoint blocking status

5.1 api system

The following subsections detail the HTTP functions available for the **api/system** service.

- 5.1.1 api system info
- 5.1.2 api system status
- 5.1.3 api system restart
- 5.1.4 api system caps

5.1.1 api system info

The **/api/system/info** function provides basic information on the device: type, serial number, firmware version, etc. The function is available in all device types regardless of the set access rights.

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes the following information on the device:

Parameter	Description
variant	Model name (version)
serialNumber	Serial number
hwVersion	Hardware version
swVersion	Firmware version
buildType	Firmware build type (alpha, beta, or empty value for official versions)
deviceName	Device name set in the configuration interface on the **Services / Web Server** tab

Example:

```
GET /api/system/info
{
  "success" : true,
  "result" : {
    "variant" : "2N Helios IP Vario",
    "serialNumber" : "08-1860-0035",
    "hwVersion" : "535v1",
    "swVersion" : "2.10.0.19.2",
    "buildType" : "beta",
    "deviceName" : "2N Helios IP Vario"
  }
}
```

5.1.2 api system status

The /api/system/status function returns the current intercom status.

The function is part of the System service and the user must be assigned the System Control privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The GET or POST method can be used for this function.

The function has no parameters.

The reply is in the application/json format and includes the current device status.

Parameter	Description
systemTime	Device real time in seconds since 00:00 1.1.1970 (unix time)
upTime	Device operation time since the last restart in seconds

Example:

```
GET /api/system/status
{
  "success" : true,
  "result" : {
    "systemTime" : 1418225091,
    "upTime" : 190524
  }
}
```

5.1.3 api system restart

The /api/system/restart restarts the intercom.

The function is part of the System service and the user must be assigned the System Control privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The GET or POST method can be used for this function.

The function has no parameters.

The reply is in the application/json format and includes no parameters.

Example:

```
GET /api/system/restart
{
  "success" : true
}
```

5.1.4 api system caps

The /api/system/caps function is used for sending information to 2N® Access Commander on a change in the list of available functions of the device.

The GET method can be used for this function.

The reply is in the application/json format and includes a set of device info.

```
{   "success":true,
    "result":{
     "options":{
         "codecG722":"active,licensed",
         "codecG729":"active",
         "codecL16":"active",
         "audioLoopTest":"active,licensed",
         "noiseDetection":"active,licensed",
         "userSounds":"active,licensed",
         "adaptiveVolume":"active",
         "antiHowling":"active",
         "keyBeep":"active",
         "camera":"active",
         "video":"active",
         "cameraPtz":"active,licensed",
         "motionDetection":"active,licensed",
         "encH264":"active",
         "encH263":"active",
         "encMpeg4":"active",
         "encJpeg":"active",
         "decH264":"active",
         "angelcam":"active",
         "phone":"active",
         "phoneVideo":"active",
         "phoneVideoOut":"active",
         "sips":"active,licensed",
         "srtp":"active,licensed",
         "callAnswerMode":"active",
         "doorOpenCallback":"active",
         "rtspServer":"active,licensed",
         "rtspClient":"active,licensed",
         "audioMulticast":"active",
         "smtpClient":"active,licensed",
         "ftpClient":"active,licensed",
         "onvif":"active,licensed",
         "snmp":"active,licensed",
         "tr069":"active,licensed",
         "knocker":"active",
         "my2n":"active",
         "informacast":"active,licensed",
         "autoProv":"active,licensed",
         "httpApi":"active,licensed",
         "eap":"active,licensed",
         "eapMd5":"active,licensed",
         "eapTls":"active,licensed",
         "vpn":"active",
         "userVpn":"active",
         "rioManager":"active,licensed",
         "siteChannel":"active",
         "localCalls":"active",
         "switches":"active",
```

```
          "advancedSwitches":"active,licensed",
          "switchUserCodes":"active",
          "securedInput":"active",
          "rexInput":"active",
          "tamperInput":"active",
          "doorSensor":"active",
          "keypad":"active",
          "buttons":"active",
          "liftControl":"active,licensed",
          "limitFailedAccess":"active,licensed",
          "silentAlarm":"active,licensed",
          "scrambleKeypad":"active,licensed",
          "tamperBlockSwitch":"active,licensed",
          "antiPassback":"active,licensed",
          "dir":"active",
          "dirDeputy":"active",
          "dirPhoto":"active",
          "automation":"active,licensed",
          "licDownload":"active",
          "profiles":"active",
          "licensing":"active",
          "accessControl":"active",
          "doorControl":"active",
          "nfc":"active,licensed",
          "vbus":"active",
          "vbusExtenders":"active",
          "cardReader":"active",
          "fpReader":"active",
          "bleReader":"active",
          "wiegand":"active",
          "powerManager":"active",
          "audioInput":"active",
          "lightSensor":"active",
          "irLed":"active",
          "backlight":"active",
          "backlightDayNight":"active",
          "display":"active"
       }

   }


}
```

5.2 api firmware

The following subsections detail the HTTP functions available for the **api/firmware** service.

- 5.2.1 api firmware
- 5.2.2 api firmware apply
- 5.2.3 api firmware reject

5.2.1 api firmware

The **api/firmware** function uploads the firmware file for upgrade/downgrade.
Methods

- PUT

Services and Privileges

- Services: System API
- Privileges: System Access Control

Request PUT

The request contains a file in **multipart/form-data**.

Table 1. Request Parameters

Parameter	Mandatory	Expected Values	Default Value	Description
blob-fw	Yes	Valid firmware binary file	-	Firmware file

Example of a PUT Request

```
http://192.168.1.1/api/firmware
```

Response to PUT

The response is in the **application/json** format. The response contains the **success** and **result** keys. The **result** value contains various keys described in the table below.

Table 2. Response JSON Keys

Key	Typical Returned Values	Description
fileId	Random identifier (8 HEX characters)	Contains a random identifier of the uploaded firmware file. The identifier must be used to confirm the uploaded firmware using **api/firmware/apply** or to reject the uploaded firmware using **api/firmware/reject**.
version	String with version identification **major.minor.patch.build.id**	Contains version identification of the uploaded firmware file.
downgrade	`true` or `false`	This flag is `true` if the uploaded firmware has a lower version than the current firmware in the device.
note	String with escaped characters	Contains an upgrade message for the uploaded firmware (e.g. warning about major changes).

Example of a Response to PUT


```
{ "success" : true, "result" : { "fileId" : "7d6adf16", "version" : "2.3
2.4.41.2", "downgrade" : false, "note" : "EN:\r\nVER=2.20.0\r\nSome
changes associated with the downgrade to a lower version result in a
loss of original settings in a certain part of configuration.\r\n\r\n*
All the cards installed in the **Directory \/ Access cards** menu are
moved to the **Directory \/ Users** menu as new users upon firmware
upgrade. Each user is automatically named as !Visitor #n, where n gives
the user number in the list. This change is irreversible upon downgrade.
\r\n* Service cards are now available in the **Hardware \/ Card
reader** menu.\r\n* All the user access ... ... ...
\u043F\u0440\u043E\u0444\u0438\u043B\u0435\u043C
\u043F\u043E\u043B\u044C\u0437\u043E\u0432\u0430\u0442\u0435\u043B\u044F
.\r\n\r\n" } }
```


The following specific error codes may be returned:

- `Error code 12`
  - param = "blob-fw"
  - description = "invalid parameter value"

- The uploaded firmware file does not match the requirements (invalid file, firmware for a different device...)

- `Error code 19`
  - description = "file version is lower than the required minimum"
  - The uploaded firmware file has a lower version than the minimum version allowed for the device.

> **Note**
>
> The device does not reply to requests to upload another firmware version when the previous firmware file is present. Use **api/firmware/reject** to reject the previous firmware first and then upload another firmware version. The uploaded firmware file is automatically rejected in 5 minutes if not applied.

5.2.2 api firmware apply

The **api/firmware/apply** function confirms the uploaded firmware file and performs device upgrade/downgrade.

Methods

- GET
- POST

Services and Privileges

- Services: System API
- Privileges: System Access Control

Request PUT

The request contains a file in **URL**.

Table 1. Request Parameters

Parameter	Mandatory	Expected Values	Default Value	Description
fileId	Yes	Firmware file identifier	-	This parameter has to correspond to the identifier of the currently uploaded firmware file.

Example of a GET or POST Request

```
http://192.168.1.1/api/firmware/apply?fileId=7d6adf16
```

Response to GET or POST

The response is in the **application/json** format. The response contains **success**. If success is `true`, the firmware is applied and the device is upgraded/downgraded.

Example of a Response to GET or POST

```
{ "success" : true }
```

The following specific error codes may be returned:

- `Error code 12`
  - parameter = "fileId"
  - description = "invalid parameter"
  - The file identifier is invalid (e.g. contains non-HEX characters).
- `Error code 14`
  - description = "new firmware not found"
  - There is no firmware file uploaded with such a fileId.

5.2.3 api firmware reject

The **api/firmware/reject** function rejects the uploaded firmware file.
Methods

- GET
- POST

Services and Privileges

- Services: System API
- Privileges: System Access Control

Request PUT

The request contains a file in URL.

Table 1. Request Parameters

Parameter	Mandatory	Expected Values	Default Value	Description
fileId	Yes	Firmware file identifier	-	This parameter has to correspond to the identifier of the currently uploaded firmware file.

Example of a GET or POST Request

```
http://192.168.1.1/api/firmware/reject?fileId=7d6adf16
```

Response to GET or POST

The response is in the application/json format. The response contains success. If success is `true`, the firmware is rejected and it is possible to upload a new firmware file using api/firmware.

Example of a Response to GET or POST

```
{ "success" : true }
```

The following specific error codes may be returned:

- `Error code 12`
- parameter = "fileId"
- description = "invalid parameter"
- The file identifier is invalid (e.g. contains non-HEX characters).
- `Error code 14`
- description = "new firmware not found"
- There is no firmware file uploaded with such fileId.

Note

- The device does not reply to the **api/firmware** requests to upload another firmware version when the previous firmware file is present. Use **api/firmware/reject** to reject the previous firmware first and then upload another firmware version. The uploaded firmware file is automatically rejected in 5 minutes if not applied.

5.3 api config

The following subsections detail the HTTP functions available for the **api/config** service.

- 5.3.1 api config
- 5.3.2 api config factoryreset
- 5.3.3 api config holidays

5.3.1 api config

The **/api/config** function helps you upload or download device configuration.

The function is part of the **System** service and the user must be assigned the **System Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

Use the **GET** or **POST** method for configuration download and **PUT** method for configuration upload.

Request parameters for **PUT**:

Parameter	Description
blob-cfg	Mandatory parameter including device configuration (XML)

No parameters are defined for the GET/POST methods.

For configuration download, the reply is in the **application/xml** format and contains a complete device configuration file.

The **/api/config** function using the **PUT** method uploads configuration with a delay of approx. 15 s; do not reset or switch off the intercom during this interval.

Example:

```
GET /api/config
<?xml version="1.0" encoding="UTF-8"?>
<!--
        Product name: 2N IP Vario
       Serial number: 08-1860-0035
    Software version: 2.10.0.19.2
    Hardware version: 535v1
  Bootloader version: 2.10.0.19.1
             Display: No
         Card reader: No
-->
<DeviceDatabase Version="4">
<Network>
    <DhcpEnabled>1</DhcpEnabled>
    ...
    ...
```

For configuration upload, the reply is in the **application/json** format and includes no other parameters.

Example:

```
PUT /api/config
{
  "success" : true
}
```

Caution

- User positions are cancelled in the directory in version 2.24. Thus, download the current configuration, make the required changes and then upload the configuration to update the directory.
- Should you fail to keep the instructions above, data may get lost.

5.3.2 api config factoryreset

The **/api/config/factoryreset** function resets the factory default values for all the intercom parameters. This function is equivalent to the function of the same name in the System / Maintenance / Default setting section of the configuration web interface.

The function is part of the **System** service and the user must be assigned the **System Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes no parameters.

The **/api/config/factoryreset** function resets the intercom factory values with a delay of approx. 15 s; do not reset or switch off the intercom during this interval.

Example:

```
GET /api/config/factoryreset
{
  "success" : true
}
```

5.3.3 api config holidays

The **/api/config/holidays** function can be used to get/set the bank holidays list.

The **GET** or **PUT** method can be used for this function.

No parameters are defined for the **GET** method.

Request parameters for **PUT** method:

Parameter	Description
blob-json	Mandatory parameter containing definition of bank holidays (JSON)

The reply of the **GET** method is in the **application/json** format and contains an array of bank holidays. The dates are formatted as DD/MM[/YYYY], where the year is specified only if the holiday is valid for the particular year only.

GET /api/config/holidays

```
{ "success" : true, "result" : { "dates": [ "01\/01", "24\/12", "01\/04\
/2018" ] } }
```

The **PUT** method JSON format is the same format as a result of the **GET** method.

```
{ "dates": [ "01\/01", "24\/12", "01\/04\/2018" ] }
```

The reply of the **PUT** method is in the **application/json** format and contains no other parameters.

PUT /api/config/holidays

```
{ "success": true
}
```

5.4 api switch

The following subsections detail the HTTP functions available for the **api/switch** service.

- 5.4.1 api switch caps
- 5.4.2 api switch status
- 5.4.3 api switch ctrl

5.4.1 api switch caps

The **/api/switch/caps** function returns the current switch settings and control options. Define the switch in the optional **switch** parameter. If the **switch** parameter is not included, settings of all the switches are returned.

The function is part of the **Switch** service and the user must be assigned the **Switch Monitoring** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
switch	Optional switch identifier (typically, 1 to 4)

The reply is in the **application/json** format and includes a switch list (**switches**) including current settings. If the **switch** parameter is used, the **switches** field includes just one item.

Parameter	Description
switch	Switch Id (1 to 4)
enabled	Switch control enabled in the configuration web interface
mode	Selected switch mode (**monostable**, **bistable**)
switchOnDuration	Switch activation time in seconds (for monostable mode only)
type	Switch type (**normal**, **security**)

Example:

```
GET /api/switch/caps
{
  "success" : true,
  "result" : {
    "switches" : [
      {
        "switch" : 1,
        "enabled" : true,
        "mode" : "monostable",
        "switchOnDuration" : 5,
        "type" : "normal"
      },
      {
        "switch" : 2,
        "enabled" : true,
        "mode" : "monostable",
        "switchOnDuration" : 5,
        "type" : "normal"
      },
      {
        "switch" : 3,
        "enabled" : false
      },
      {
        "switch" : 4,
        "enabled" : false
      }
    ]
  }
}
```

5.4.2 api switch status

The **api/switch/status** function returns the current switch statuses.

Service and Privileges Groups

- Service group is Switch.
- Privileges group is Switch Access Control.

Methods

- GET
- POST

Request

The request contains parameters in the URL (or in the **application/x-www-form-urlencoded** format when POST is used).

Table 1. Request Parameters

Parameter Name	Mandatory	Expected Values	Default Value	Description
switch	No	Integer defining a switch (typically 1 to 4)	-	Defines which switch status will be returned. **api/switch/caps** can be used for obtaining the number of switches of a particular device. The status of switches is returned if this parameter is omitted.

Example of a Request

```
URL: https://192.168.1.1/api/switch/status?switch=1
```

Response

The success response is in the **application/json** format. It contains two JSON keys `success` and `result`, which contains the key `switches` (status information on individual switches are in an Array of one to four elements).

Table 2. Response switches JSON Keys

Key	Typical Returned Values	Description
switch	Integer (typically 1 to 4)	Defines to which switch the status is related.
active	`true` or `false`	Defines the current state of the switch (`true` - the switch is activated, `false` - the switch is deactivated).
locked	`true` or `false`	Defines whether the switch is locked or not (`true` - the switch is locked in deactivated position and cannot be operated, `false` - the switch is unlocked and can be operated normally). Locking has the priority over holding the switch - i.e. when the switch is simultaneously locked and held, it is deactivated and cannot be operated.

Key	Typical Returned Values	Description
held	`true` or `false`	Defines whether the switch is held or not (`true` - the switch is held in activated position and cannot be operated, `false` - the switch is released and can be operated normally). Locking has the priority over holding the switch - i.e. when the switch is simultaneously locked and held, it is deactivated and cannot be operated.

Example of a Response


```
{ "success": true, "result": { "switches": [ { "switch": 1, "active": tr
ue, "locked": false, "held": true }, { "switch": 2, "active": true, "loc
ked": false, "held": false }, { "switch": 3, "active": false, "locked":
true, "held": true }, { "switch": 4, "active": false, "locked": true, "h
eld": false } ] } }
```


There may occur various errors (e.g. missing mandatory parameter). Errors are returned in json with a response code 200.

5.4.3 api switch ctrl

The **/api/switch/ctrl** is used for control of switches.

Service and Privileges Groups

- Service group is Switch.
- Privileges group is Switch Access Control.

Methods

- GET
- POST

Request

The request contains parameters in the URL (or in the **application/x-www-form-urlencoded** format when POST is used).

Table 1. Request Parameters

Parameter Name	Mandatory	Expected Values	Default Value	Description
switch	Yes	Integer defining a switch (typically 1 to 4)	-	Defines which switch will be controlled. **api/switch/caps** can be used for obtaining the number of switches of a particular device.
action	Yes	String defining the command	-	Defines which command will be applied to the switch. The available commands are: • `on` - activate switch • `off` - deactivate switch • `trigger` - activate monostable switch, toggle the state of bistable switch • `lock` - lock switch (locked switch is deactivated and cannot be operated) • `unlock` - unlock switch (allow normal operation) • `hold` - hold switch activated (held switch is activated and cannot be operated), if the switch is locked and held together it is deactivated • `release` - release the switch from being held (allow normal operation)
response	No	String defining a text that is to be returned instead of standard JSON response	-	The device will return the text specified in this parameter instead of a standard JSON response.

Example of a Request

```
URL: https://192.168.1.1/api/switch/ctrl?
switch=4&action=trigger&response=TEST
```

Response

The success response is in the **application/json** format (unless a custom text response is defined in the parameter `response`).

Table 2. Response JSON Keys

Key	Typical Returned Values	Description
success	`true` or `false`	When a command was performed successfully, the success value is `true`, and it is `false` when the requested switch state could not be reached (e.g. the switch is locked and the requested state is activated switch).

Example of a Response

```
{ "success": true }
```

Additional error information is contained in the response when the success is `false`. Error code 14 "action failed" is returned when the requested result could not be achieved (e.g. when the switch is locked and `action=on` is requested). A command to change the operation type (i.e. held, locked) will always succeed since the operation can be changed all the time except when the switch is disabled (a device will return error 14 to all commands when the switch is disabled).

5.5 api io

The following subsections detail the HTTP functions available for the **api/io** service.

- 5.5.1 api io caps
- 5.5.2 api io status
- 5.5.3 api io ctrl

5.5.1 api io caps

The **/api/io/caps** function returns a list of available hardware inputs and outputs (ports) of the device. Define the input/output in the optional **port** parameter. If the **port** parameter is not included, settings of all the inputs and outputs are returned.

The function is part of the **I/O** service and the user must be assigned the **I/O Monitoring** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
Port	Optional input/output identifier

The reply is in the **application/json** format and includes a port list (**ports**) including current settings. If the **port** parameter is used, the **ports** field includes just one item.

Parameter	Description
port	Input/output identifier
type	Type (**input** – for digital inputs, **output** – for digital outputs)

Example:

```
GET /api/io/caps
{
  "success" : true,
  "result" : {
    "ports" : [
      {
        "port" : "relay1",
        "type" : "output"
      },
      {
        "port" : "relay2",
        "type" : "output"
      }
   ]
  }
}
```

5.5.2 api io status

The **/api/io/status** function returns the current statuses of logic inputs and outputs (ports) of the device. Define the input/output in the optional **port** parameter. If the **port** parameter is not included, statuses of all the inputs and outputs are returned.

The function is part of the **I/O** service and the user must be assigned the **I/O Monitoring** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
port	Optional input/output identifier. Use also **/api/io/caps** to get identifiers of the available inputs and outputs.

The reply is in the **application/json** format and includes a port list (**ports**) including current settings (**state**). If the **port** parameter is used, the **ports** field includes just one item.

Example:

```
GET /api/io/status
{
  "success" : true,
  "result" : {
    "ports" : [
      {
        "port" : "relay1",
        "state" : 0
      },
      {
        "port" : "relay2",
        "state" : 0
      }
    ]
  }
}
```

5.5.3 api io ctrl

The /api/io/ctrl function controls the statuses of the device logic outputs. The function has two mandatory parameters: port, which determines the output to be controlled, and action, defining the action to be executed over the output (activation, deactivation).

The function is part of the I/O service and the user must be assigned the I/O Control privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The GET or POST method can be used for this function.

Request parameters:

Parameter	Description
port	Mandatory I/O identifier. Use also /api/io/caps to get the identifiers of the available inputs and outputs.
action	Mandatory action defining parameter (on – activate output, log. 1, off – deactivate output, log. 0)
response	Optional parameter modifying the intercom response to include the text defined here instead of the JSON message.

The reply is in the application/json format and includes no parameters.

Example:

```
GET /api/io/ctrl?port=relay1&action=on
{
  "success" : true
}
```

If the response parameter is used, the reply does not include the json messages; the server returns a text/plain reply with the specified text (which can be empty).

Example:

```
GET /api/io/ctrl?port=relay1&action=on&response=text
text
```

```
GET /api/io/ctrl?port=relay1&action=on&response=
```

5.6 api phone

The following subsections detail the HTTP functions available for the **api/phone** service.

- 5.6.1 api phone staus

5.6.1 api phone staus

The **/api/phone/status** functions helps you get the current statuses of the device SIP accounts.

The function is part of the **Phone/Call** service and the user must be assigned the **Phone /Call Monitoring** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
account	Optional SIP account identifier (1 or 2). If the parameter is not included, the function returns statuses of all the SIP accounts.

The reply is in the **application/json** format and includes a list of device SIP accounts (**accounts**) including current statuses. If the **account** parameter is used, the **accounts** field includes just one item.

Parameter	Description
account	Unique SIP account identifier (1 or 2)
sipNumber	SIP account telephone number
registered	Account registration with the SIP Registrar
registerTime	Last successful registration time in seconds since 00:00 1.1.1970 (unix time)

Example:

```
GET /api/phone/status
{
  "success" : true,
  "result" : {
    "accounts" : [
      {
        "account" : 1,
        "sipNumber" : "5046",
        "registered" : true,
        "registerTime" : 1418034578
      },
      {
        "account" : 2,
        "sipNumber" : "",
        "registered" : false
      }
    ]
  }
}
```

5.7 api call

The following subsections detail the HTTP functions available for the **api/call** service.

- 5.7.1 api call status
- 5.7.2 api call dial
- 5.7.3 api call answer
- 5.7.4 api call hangup

5.7.1 api call status

The **/api/call/status** function helps you get the current states of active telephone calls. The function returns a list of active calls including parameters.

The function is part of the **Phone/Call** service and the user must be assigned the **Phone /Call Monitoring** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
session	Optional call identifier. If the parameter is not included, the function returns statuses of all the active calls.

The reply is in the **application/json** format and includes a list of active calls (**sessions**) including their current states. If the **session** parameter is used, the **sessions** field includes just one item. If there is no active call, the **sessions** field is empty.

Parameter	Description
session	Call identifier
direction	Call direction (**incoming**, **outgoing**)
state	Call state (**connecting**, **ringing**, **connected**)

Example:

```
GET /api/call/status
{
  "success" : true,
  "result" : {
    "sessions" : [
      {
        "session" : 1,
        "direction" : "outgoing",
        "state" : "ringing"
      }
    ]
  }
```

5.7.2 api call dial

The **/api/call/dial** function initiates a new outgoing call to a selected phone number or sip uri.

The function is part of the **Phone/Call** service and the user must be assigned the **Phone /Call Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
number	Mandatory parameter specifying the destination phone number or sip uri

The reply is in the **application/json** format and includes information on the outgoing call created.

Parameter	Description
session	Call identifier, used, for example, for call monitoring with **/api/call/status** or call termination with **/api/call/hangup**

Example:

```
GET /api/call/dial?number=sip:1234@10.0.23.194
{
  "success" : true,
  "result" : {
    "session" : 2
  }
}
```

5.7.3 api call answer

The **/api/call/answer** function helps you answer an active incoming call (in the **ringing** state).

The function is part of the **Phone/Call** service and the user must be assigned the **Phone /Call Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
session	Active incoming call identifier

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/call/answer?session=3
{
  "success" : true
}
```

5.7.4 api call hangup

The **/api/call/hangup** helps you hang up an active incoming or outgoing call.

The function is part of the **Phone/Call** service and the user must be assigned the **Phone /Call Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
session	Active incoming/outgoing call identifier
reason	End call reason: **normal** – normal call end (default value) **rejected** – call rejection signalling **busy** – station busy signalling

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/call/hangup?session=4
{
  "success" : true
}
```

5.8 api camera

The following subsections detail the HTTP functions available for the **api/camera** service.

- 5.8.1 api camera caps
- 5.8.2 api camera snapshot

5.8.1 api camera caps

The **/api/camera/caps** function returns a list of available video sources and resolution options for JPEG snapshots to be downloaded via the **/api/camera/snapshot** function.

The function is part of the **Camera** service and the user must be assigned the **Camera Monitoring** privilege for authentication if required.

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes a list of supported resolutions of JPEG snapshots (**jpegResolution**) and a list of available video sources (**sources**), which can be used in the **/api/camera/snapshot** parameters.

Parameter	Description
width, height	Snapshot resolution in pixels
source	Video source identifier

Example:

```
GET /api/camera/caps
{
  "success" : true,
  "result" : {
    "jpegResolution" : [
      {
        "width" : 160,
        "height" : 120
      },
      {
        "width" : 176,
        "height" : 144
      },
      {
        "width" : 320,
        "height" : 240
      },
      {
        "width" : 352,
        "height" : 272
      },
      {
        "width" : 352,
        "height" : 288
      },
      {
        "width" : 640,
        "height" : 480
      }
    ],
    "sources" : [
      {
        "source" : "internal"
      },
      {
        "source" : "external"
      }
    ]
  }
}
```

5.8.2 api camera snapshot

The **/api/camera/snapshot** function helps you download images from an internal or external IP camera connected to the intercom. Specify the video source, resolution and other parameters.

The function is part of the **Camera** service and the user must be assigned the **Camera Monitoring** privilege for authentication if required.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
width	Mandatory parameter specifying the horizontal resolution of the JPEG image in pixels
height	Mandatory parameter specifying the vertical resolution of the JPEG image in pixels. The snapshot height and width must comply with one of the supported options (see **api/camera/caps**).
source	Optional parameter defining the video source (**internal** – internal camera, **external** – external IP camera). If the parameter is not included, the default video source included in the Hardware / Camera / Common settings section of the configuration web interface is selected.
fps	Optional parameter defining the frame rate. If the parameter is set to >= 1, the intercom sends images at the set frame rate using the **http server push** method.
time	Optional parameter defining the snapshot time in the intercom memory. The time values must be within the intercom memory range: <-30, 0> seconds. When this parameter is used together with the **fps** parameter, the **fps** parameter is ignored and function returns only a single frame.

The reply is in the **image/jpeg** or **multipart/x-mixed-replace** (pro fps >= 1) format. If the request parameters are wrong, the function returns information in the **application /json** format.

Example:

```
 GET /api/camera/snapshot?width=640&height=480&source=internal

 # following command returns a frame which was captured 5 seconds
before the command was executed
 GET /api/camera/snapshot?width=640&height=480&source=internal&time=-5
```

Caution

- 2N® IP Style and other high resolution supporting 2N IP intercoms return a static image (i.e. unless the fps parameter is specified) in the maximum resolution of 1280 x 960. In case a higher resolution is required, the 1280 x 960 resolution is still returned.

5.9 api display

The following subsections detail the HTTP functions available for the **api/display** service.

- 5.9.1 api display caps
- 5.9.2 api display image

5.9.1 api display caps

The **/api/display/caps** function returns a list of device displays including their properties. Use the function for display detection and resolution.

The function is part of the **Display** service and the user must be assigned the **Display Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes a list of available displays (**displays**).

Parameter	Description
display	Display identifier
resolution	Display resolution in pixels

Example:

```
GET /api/display/caps
{
  "success" : true,
  "result" : {
    "displays" : [
      {
        "display" : "internal",
        "resolution" : {
          "width" : 320,
          "height" : 240
        }
      }
    ]
  }
}
```

5.9.2 api display image

2N ® IP Verso

The **/api/display/image** function helps you modify the content to be displayed: upload a GIF / JPEG / BMP image to or delete an earlier uploaded image from the display.

The function is part of the **Display** service and the user must be assigned the **Display Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **PUT** or **DELETE** method can be used for this function: **PUT** helps upload an image to the display, **DELETE** helps delete an uploaded image from the display.

PUT method

Request parameters:

Parameter	Description
display	Mandatory display identifier. Find the value in Hardware/Extending modules/Module name or using **/api/display/caps**.
blob-image	Mandatory parameter containing a JPEG / BMP / PNG image with 214 x 240 display resolution (refer to **/api/display/caps**). The parameter is applied only if the **PUT** method is used. The request may contain just one parameter: blob-image or blob-video.

Parameter	Description
blob-video	Mandatory parameter containing an MPEG4 / H264 video of the maximum duration of 60 s, maximum of 15 fps and resolution of 214 x 240 pixels. The request may contain just one parameter: blob-image or blob-video.
duration	Optional parameter. Image display / video playing time. The parameter is set in milliseconds.
repeat	Optional parameter. Video playing repetition count. The parameter applies to video only.

There are two ways how to display an image: as a notification or overlay. Notifications are displayed for a predefined period of time and automatically disappear after the timeout. Overlays keep displayed until replaced with another image or removed by the user.

If the HTTP request does not include any of the above mentioned parameters, the overlay mode is used, i.e. the image is displayed for an indefinite period of time. If both the parameters are included, the display is ended by the event that happens first.

The reply is in the **application/json** format and includes no parameters.

Image parameters:

Model	Image size	Supported formats
2N® IP Verso	214 x 240 pixels	JPEG (recommended), BMP, PNG

Note

- The supported JPEG format is JPEG Baseline (non-progressive encoding).

Example:

```
  api/display/image?display=ext1&duration=30000
{
  "success" : true
}
```

Video parameters:

Model	Video size	Supported formats
2N® IP Verso	214 x 240 pixels	MPEG4 / H264: Baseline profile, up to 5.2 level

Example:

```
  api/display/image?display=ext1&repeat=5
 {
  "success" : true
}
```

DELETE method

Parameter	Description
display	Mandatory display identifier. Find the value in Hardware/Extending modules/Module name or using /api/display/caps.

Example:

```
DELETE /api/display/image?display=ext1
{
  "success" : true
}
```

2N ® IP Vario

The /api/display/image function helps you modify the content to be displayed: upload a GIF / JPEG / BMP image to or delete an earlier uploaded image from the display.

Note

- The function is available only if the standard display function is disabled in the Hardware / Display section of the configuration web interface.

The function is part of the Display service and the user must be assigned the Display Control privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The PUT or DELETE method can be used for this function: PUT helps upload an image to the display, DELETE helps delete an uploaded image from the display.

Request parameters:

Parameter	Description
display	Mandatory display identifier (**internal**).
blob-image	Mandatory parameter including an image in the supported format with display resolution (see **/api/display/caps**). The parameter is applied only if the **PUT** method is used.

The reply is in the **application/json** format and includes no parameters.

Image parameters:

Model	Image size	Supported formats
2N® IP Vario	320 x 240 pixels	JPEG (recommended), GIF, BMP

> ⚠ **Caution**
>
> The supported JPEG format is JPEG Baseline (non-progressive encoding).

Example:

```
DELETE /api/display/image?display=internal
{
  "success" : true
}
```

- 5.9.1 api display image examples

5.9.1 api display image examples

The below-mentioned examples help sending data from the control application to the 2N® IP Verso and 2N® IP Vario displays.

An image can be displayed either as a notification or overlay. 2N® IP Verso can display images in either way, 2N® IP Vario can only display notifications. Notifications are displayed for a pre-defined time and disappear automatically after this timeout. Overlays keep displayed until replaced with another image or removed by the user.

The *duration* parameter gives the image/video display time in ms. The *repeat* parameter specifies the count of video repetitions and is ignored for images.

If the HTTP request does not include any of the above-mentioned parameters, the overlay mode is used, i.e. the image is displayed for an indefinite period of time. If both the parameters are included, the display is terminated by the event that happens first.

Image Loading to 2N ® IP Verso/2N ® IP Vario Display

Note

Each model supports a different image resolution.

Model	Image size	Supported formats
2N® IP Verso	214 x 240 pixels	JPEG (recommended), BMP, PNG
2N® IP Vario	320 x 240 pixels	JPEG (recommended), GIF, BMP

Request URL: https://10.27.24.15/api/display/image?display=ext1

- Request method: PUT
- Remote address: 10.27.24.15:443
- Status code: 200 OK
- Version: HTTP/1.1

Response headers (95 B)

- Server: HIP2.22.0.31.1
- Content-Type: application/json
- Content-Length: 24

Request headers (494 B)

- Host: 10.27.24.15
- User-Agent: Mozilla/5.0 (Windows NT 6.1; W...) Gecko/20100101 Firefox/56.0
- Accept: */*
- Accept-Language: cs,en-US;q=0.7,en;q=0.3
- Accept-Encoding: gzip, deflate, br
- Referer: https://10.27.24.15/apitest.html
- Content-Length: 1325
- Content-Type: multipart/form-data; boundary=...----------258852674219952
- Cookie: _ga=GA1.1.375392382.1496656977...id=GA1.1.638680516.1507547865
- Connection: keep-alive

Query string

- display: ext1

Request payload

-----------------------------258852674219952

Content-Disposition: form-data; name="blob-image"; filename="picture.png"

Content-Type: image/png


```
‰PNG
IHDR    Ö       ót
R    sRGB ®Îé    gAMA   ±üa        pHYs     Ço¨d  üIDATx^íq8@Ô•,\Oq3)»Ž4Y,$'Ô
¬ÎÔ\ƒ}@J,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$H:     }íýáIžy/@,IP
,±×L‘ ±Ó\-^b;     ŽkëÓ©í-f)ËýBÎLç÷m.jb¶v?˜m|;éJBÎK0‹›a÷/›dY¸Z$Á9+Áá[CC/´)
é68[\quŽN%Á©oÂöu6—×mcš%x!§Žl¶°¦I¸Ë5céw¬“`©     •}š%¸Íi7še`»Î"Á9;ÁáÚ˜%
¸ÍbwúHI_^“—q|7Ëí—‹:+Á Áög±; ?8)Áâ‹     ÝfK_'Á÷x9FÇYL¦Ó«ÝÇ÷â©žâ|Z?ž—lââ
(\M†KJBÎú-848µ|x~-Á=r«ƒ%•·÷d±ÇY<>Ý‘x|5ž?-¸öWR‰fâ#ß”`ÜJ,’    î4^1Áá]Y,µ—
JpšO
R,;ú!{WokxÉ×®D,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL,Ä$HL
,Ä$HL,¤>>+uKé<:-     IEND®B`‚
```


```
-----------------------------258852674219952--
```

Video Loading to 2N ® IP Verso Display

Info

Model	Video size	Supported formats
2N® IP Verso	214 x 240 pixels	MPEG4 / H264: Baseline profile, level up to 5.2

Request URL: https://10.27.24.15/api/display/image?display=ext1&duration=20&repeat=3

- Request method: PUT
- Remote address: 10.27.24.15:443
- Status code: 200 OK
- Version: HTTP/1.1

Response headers (95 B)

- Server: HIP2.22.0.31.1
- Content-Type: application/json
- Content-Length: 24

Request headers (516 B)

- Host: 10.27.24.15
- User-Agent: Mozilla/5.0 (Windows NT 6.1; W...) Gecko/20100101 Firefox/56.0
- Accept: */*
- Accept-Language: cs,en-US;q=0.7,en;q=0.3
- Accept-Encoding: gzip, deflate, br
- Referer: https://10.27.24.15/apitest.html
- Content-Length: 943815
- Content-Type: multipart/form-data; boundary=---------------------------14948718218673
- Cookie: _ga=GA1.1.375392382.1496656977...id=GA1.1.638680516.1507547865
- Connection: keep-alive

Query string

- display – ext1
- duration – 20
- repeat – 3

Request payload

----------------------------14948718218673

Content-Disposition: form-data; name="blob-video"; filename="2N_intro.mp4"

Content-Type: video/mp4

```
ftypmp42  isomiso2avc1mp41   free O!mdat  ·ÜEéH·–, #îx264 - core 148
r2708 86b7198 - H.264/MPEG-4 AVC codec - Copyleft 2003-2016 - http://www
.videolan.org/x264.html - options: cabac=0 ref=2 deblock=0:0:0
analyse=0x1:0x111 me=hex subme=6 psy=1 psy_rd=1.00:0.00 mixed_ref=1
me_range=16 chroma_me=1 trellis=1 8x8dct=0 cqm=0 deadzone=21,11
fast_pskip=1 chroma_qp_offset=-2 threads=6 lookahead_threads=1
sliced_threads=0 nr=0 decimate=1 interlaced=0 bluray_compat=0
constrained_intra=0 bframes=0 weightp=0 keyint=150 keyint_min=15
scenecut=40 intra_refresh=0 rc_lookahead=30 rc=crf mbtree=1 crf=22.0
qcomp=0.60 qpmin=0 qpmax=69 qpstep=4 vbv_maxrate=20000
vbv_bufsize=25000 crf_max=0.0 nal_hrd=none filler=0 ip_ratio=1.40 aq=1:
1.00 €  eˆ„    ®C‡Ü!q@ CzNbyRÔ~ˆ$]´í·Fb0~hM=>?„´mßâµWj«¸z”ÄO”#Ü~éÂ6Ë-Ó
üP•?n?é¦žš{–ü7-«“b»ž¶˜G
LZ.˜üßÓý×\’žjo¸…zxk&+ýGkü¸3 ˆ|p«^¸UzÖXÉž¤€p!>lÍMfÓo_„0„)Îl,Ë?_?|{>Îl&
/Í÷~»T<vW
/}     e‡•¶ßy9šziý-$‰_Á+{]]u×]wÜ"ý9§ü$l7zë®ë®ë®úë®ë®ë®ë®ë®ë®xxCî‰
®ë®ë®:OäyŽŠ{Osó«‘u]u×S·,\7·p°Ó–X    ]`§Äƒüš§/ó Gi...
```

----------------------------14948718218673--

5.10 api log

The following subsections detail the HTTP functions available for the **api/log** service.

- 5.10.1 api log caps
- 5.10.2 api log subscribe
- 5.10.3 api log unsubscribe
- 5.10.4 api log pull

5.10.1 api log caps

The **/api/log/caps** function returns a list of supported event types that are recorded in the device. This list is a subset of the full event type list below:

Event type	Description	Note
DeviceState	Signals a system event generated at device state changes.	
AudioLoopTest	Signals performance and result of an automatic audio loop test.	with a valid **Enhanced Audio** licence key only
MotionDetected	Signals motion detection via a camera.	for camera-equipped models only
NoiseDetected	Signals increased noise level detection.	for microphone /microphone input equipped models only
KeyPressed	Signals pressing of a speed dial button, numeric keypad button, display touch or the Bluetooth authentication initializing key.	
KeyReleased	Signals releasing of a speed dial/numeric keypad button.	
CodeEntered	Signals entering of a user code via the numeric keypad.	

Event type	Description	Note
		for numeric keypad equipped models only
CardEntered	Signals tapping of an RFID card on the card reader.	for RFID card reader equipped models only
InputChanged	Signals a logic input state change.	
OutputChanged	Signals a logic output state change.	
SwitchStateChanged	Signals a switch 1–4 state change.	
CallStateChanged	Signals a setup/end/change of the active call state.	
RegistrationStateChanged	Signals a SIP server registration state change.	
TamperSwitchActivated	Signals tamper switch activation.	for tamper switch equipped models only
UnauthorizedDoorOpen	Signals unauthorised door opening.	for digital input equipped models only
DoorOpenTooLong	Signals excessively long door opening, or door closing failure within the timeout.	for digital input equipped models only
LoginBlocked	Signals temporary blocking of login to the web interface.	
UserAuthenticated	Signals user authentication and subsequent door opening.	
CardHeld	Signals RFID card tapping and holding for more than 4 s.	

Event type	Description	Note
SilentAlarm	Signals silent alarm activation.	
ApLockStateChange	Signals a lock state change.	
RexActivated	Signals REX departure button activation.	
AccessLimited	Signals user rejection.	
PairingStateChanged	Signals pairing with a Bluetooth interface.	
SwitchesBlocked	Signals lock blocking by tamper switch activation.	
FingerEntered	Signals that a finger has been swiped across the biometric reader.	
MobKeyEntered	Signals Bluetooth reader authentication.	
DoorStateChanged	Signals a door state change.	
UserRejected	Signals user authentication rejection.	
DisplayTouched	Signals display touch.	for 2N® IP Verso and 2N® Access Unit models only
CapabilitiesChanged	Signals a change in available functions.	

The function is part of the **Logging** service and requires no special user privileges.

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format:

Parameter	Type	Description
events	array	Array of strings including a list of supported event types

Example:

```
GET /api/log/caps
{
  "success" : true,
  "result" : {
    "events" : [
      "KeyPressed",
      "KeyReleased",
      "InputChanged",
      "OutputChanged",
      "CardEntered",
      "CallStateChanged",
      "AudioLoopTest",
      "CodeEntered",
      "DeviceState",
      "RegistrationStateChanged"
    ]
  }
}
```

5.10.2 api log subscribe

The **/api/log/subscribe** function helps you create a subscription channel and returns a unique identifier to be used for subsequent dialling of the **/api/log/pull** or **/api/log/unsubscribe** function.

Each subscription channel contains an event queue of its own. All the new events that match the channel filter (**filter** parameter) are added to the channel queue and read using the **/api/log/pull** function.

At the same time, the device keeps the event history queue (last 10000 events) in its internal memory. The event history queue is empty by default.

Use the **include** parameter to specify whether the channel queue shall be empty after restart (storing of events occurring after the channel is opened), or be filled with all or some events from the event history records.

Use the **duration** parameter to define the channel duration if it is not accessed via **/api/log/pull**. The channel will be closed automatically when the defined timeout passes as if the **/api/log/unsubscribe** function were used.

The function is part of the **Logging** service and requires some user privileges for authentication. Unprivileged user events shall not be included in the channel queue.

Event type	Required user privileges
DeviceState	None
AudioLoopTest	None
MotionDetected	None
NoiseDetected	None
KeyPressed	Keypad monitoring
KeyReleased	Keypad monitoring
CodeEntered	Keypad monitoring
CardEntered	UID monitoring (cards/Wiegand)
InputChanged	I/O monitoring
OutputChanged	I/O monitoring
SwitchStateChanged	I/O monitoring
CallStateChanged	Call/phone monitoring
RegistrationStateChanged	Call/phone monitoring
TamperSwitchActivated	None
UnauthorizedDoorOpen	None
DoorOpenTooLong	None
LoginBlocked	None
UserAuthenticated	None
CardHeld	None

Event type	Required user privileges
SilentAlarm	None
ApLockStateChanged	None
RexActivated	None
AccessLimited	None
PairingStateChanged	None
SwitchesBlocked	None
FingerEntered	None
MobKeyEntered	None
DoorStateChanged	None
UserRejected	None
DisplayTouched	None

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Type	Mandatory	Default value	Description
include	string	No	new	Define the events to be added to the channel event queue: **new** - only new events occurring after channel creation **all** - all events recorded so far including those occurring after channel creation **-t** - all events recorded in the last **t** seconds including those occurring after channel creation (-10, e.g.)
filter	list	No	no filter	List of required event types separated with commas. The parameter is optional and if no value is entered, all available event types are transferred via the channel.
duration	uint32	No	90	Define a timeout in seconds after which the channel shall be closed automatically if no **/api/log/pull** reading operations are in progress. Every channel reading automatically extends the channel duration by the value included here. The maximum value is 3600 s.

The reply is in the **application/json** format and includes an identifier created by subscription.

Parameter	Type	Description
id	uint32	Unique identifier created by subscription

Example:

```
GET /api/log/subscribe?filter=KeyPressed,InputChanged
{
  "success" : true,
  "result" : {
    "id" : 2121013117
  }
}
```

5.10.3 api log unsubscribe

The **/api/log/unsubscribe** function helps you close the subscription channel with the given identifier. When the function has been executed, the given identifier cannot be used, i.e. all subsequent **/api/log/pull** or **/api/log/unsubscribe** calls with the same identifier will end up with an error.

The function is part of the **Logging** service and requires no special user privileges.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Type	Mandatory	Default value	Description
id	uint32	Yes	–	Identifier of the existing channel obtained by preceding dialling of **/api/log/subscribe**

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/log/unsubscribe?id=21458715
{
  "success" : true,
}
```

5.10.4 api log pull

The **/api/log/pull** helps you read items from the channel queue (subscription) and returns a list of events unread so far or an empty list if no new event is available. Larger amounts of events are pulled in batches of 128 events.

Use the **timeout** parameter to define the maximum time for the intercom to generate the reply. If there is one item at least in the queue, the reply is generated immediately. In case the channel queue is empty, the intercom puts off the reply until a new event arises or the defined timeout elapses.

The function is part of the **Logging** service and requires no special user privileges.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Type	Mandatory	Default value	Description
id	uint32	Yes	–	Identifier of the existing channel created by preceding dialling of **/api/log/subscribe**
timeout	uint32	No	0	Define the reply delay (in seconds) if the channel queue is empty. The default value 0 means that the intercom shall reply without delay.

The reply is in the **application/json** format and includes a list of events.

Parameter	Type	Description
events	array	Event object array. If no event occurs during the timeout, the array is empty.

Example:

```
GET /api/log/pull
{

  "success" : true,
  "result" : {
    "events" : [
      {
        "id" : 1,
        "tzShift" : 0,
        "utcTime" : 1437987102,
        "upTime" : 8,
        "event" : "DeviceState",
        "params" : {
          "state" : "startup"
        }
      },
      {
        "id" : 3,
        "tzShift" : 0,
        "utcTime" : 1437987105,
        "upTime" : 11,
        "event" : "RegistrationStateChanged",
        "params" : {
          "sipAccount" : 1,
          "state" : "registered"
        }
      }
    ]
  }
}
```

Events

Each event in the **events** field includes the following common information:

Parameter	Type	Description
id	uint32	Internal event record ID (32bit number, 1 after intercom restart incremented with every new event)
utcTime	uint32	Absolute event rise time (Unix Time, UTC)
upTime	uint32	Relative event rise time (seconds after intercom restart)
tzShift	int32	Difference between the local time and Coordinated Universal Time (UTC) in minutes. Add this value to utcTime to obtain the local time of event generation according to the device time zone: localTime = utcTime + tzShift * 60
event	string	Event type (KeyPressed, InputChanged, ...)
params	object	Specific event parameters

DeviceState

Signals the device state changes.

Event parameters:

Parameter	Type	Description
state	string	Signalled device state: **startup** – generated one-time after device start (always the first event ever)

Example:

```
{
  "id" : 1,
  "tzShift" : 0,
  "utcTime" : 1437987102,
  "upTime" : 8,
  "event" : "DeviceState",
  "params" : {
    "state" : "startup"
  }
}
```

AudioLoopTest

Signals performance and result of an automatic audio loop test. The **AudioLoopTest** event is only available in selected models with a valid **Enhanced Audio** licence. The event is signalled whenever the automatic test has been performed (either scheduled or manually started).

Parameter	Type	Description
result	string	Result of an accomplished text: **passed** – the test was carried out successfully, no problem has been detected. **failed** – the test was carried out, a loudspeaker/microphone problem has been detected.

Example:

```
{
 "id" : 26,
 "tzShift" : 0,
 "utcTime" : 1438073190,
 "upTime" : 9724,
 "event" : "AudioLoopTest",
 "params" : {
   "result" : "passed"
 }
}
```

MotionDetected

Signals motion detection via a camera. The event is available in camera-equipped models only. The event is generated only if the function is enabled in the intercom camera configuration.

Event parameters:

Parameter	Type	Description
state	string	Motion detector state: in – signals the beginning of the interval in which motion was detected. out – signals the end of the interval in which motion was detected.

Example:

```
{
  "id" : 2,
  "tzShift" : 0,
  "utcTime" : 1441357589,
  "upTime" : 1,
  "event" : "MotionDetected",
  "params" : {
    "state" : "in"
  }
}
```

NoiseDetected

Signals an increased noise level detected via an integrated or external microphone. The event is generated only if this function is enabled in the intercom configuration.

Event parameters:

Parameter	Type	Description
state	string	Noise detector state: **in** – signals the beginning of the interval in which noise was detected. **out** – signals the end of the interval in which noise was detected.

Example:

```
{
  "id" : 2,
  "tzShift" : 0,
  "utcTime" : 1441357589,
  "upTime" : 1,
  "event" : "NoiseDetected",
  "params" : {
    "state" : "in"
  }
}
```

KeyPressed and KeyReleased

Signals pressing **(KeyPressed)** or releasing **(KeyReleased)** of speed dial or numeric keypad buttons.

Event parameters:

Parameter	Type	Description
key	string	Pressed/released button code: **0** to **9** – numeric keypad buttons **%1**–**%150** – speed dialling buttons ***** – button with a * or phone symbol **#** – button with a # or key symbol

Example:

```
{
  "id" : 4,
  "tzShift" : 0,
  "utcTime" : 1437987888,
  "upTime" : 794,
  "event" : "KeyPressed",
  "params" : {
    "key" : "5"
  }
}
```

CodeEntered

Signals entering of a user code via the numeric keypad. The event is generated in numeric keypad equipped devices only.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit
session	string	Informs how many times the code has been entered.
direction	string	Code direction: in – arrival out – departure any – passage *Note: Set the card reader direction using the intercom configuration interface.*
code	string	User code, 1234, e.g. The code includes 2 digits at least and 00 cannot be used.
type	string	
uuid	string	User's unique ID
valid	boolean	Code validity (i.e. if the code is defined as a valid user code or universal switch code in the intercom configuration): false – invalid code true – valid code

Example:


```
{
  "id" : 128,
  "tzShift" : 0,
  "utcTime" : 1548078453,
  "upTime" : 1061,
  "event" : "CodeEntered",
  "params" : {
    "ap" : 0,
    "session" : 8,
    "direction" : "in",
    "code" : "1234",
    "type" : "user",
    "uuid" : "54877b0e-4cc3-c645-9530-6c7850f47a9c",
    "valid" : true
  }
}
```

CardEntered

Signals tapping an RFID card on the card reader. The event is generated in RFID card reader equipped devices only.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit
session		Informs how many times the card has been applied.
direction	string	RFID direction: in – arrival out – departure any – passage *Note: Set the card reader direction using the intercom configuration interface.*
reader	string	RFID card reader/Wiegand module name, or one of the following non-modular intercom model values: internal – internal card reader (2N IP intercoms) external – external card reader connected via the Wiegand interface *Note: Set the card reader name using the intercom configuration interface .*
uid	string	Unique identifier of the applied card (hexadecimal format, 6 - 16 characters depending on the card type)
uuid	string	User's unique ID
valid	boolean	Validity of the applied RFID card (if the card uid is assigned to one of the intercom users listed in the phonebook) false – invalid card true – valid card

Example:


```
{
  "id" : 60,
  "tzShift" : 0,
  "utcTime" : 1548078014,
  "upTime" : 622,
  "event" : "CardEntered",
  "params" : {
    "ap" : 0,
    "session" : 5,
    "direction" : "in",
    "reader" : "ext2",
    "uid" : "4BD9E903",
    "uuid" : "54877b0e-4cc3-c645-9530-6c7850f47a9c",
    "valid" : true
  }
}
```

InputChanged and OutputChanged

Signals a state change of the logic input **(InputChanged)** or output **(OutputChanged)**. Use the **/api/io/caps** function to get the list of available inputs and outputs.

Event parameters:

Parameter	Type	Description
port	string	I/O port name
state	boolean	Current I/O port logic state: **false** – inactive, log. 0 **true** – active, log. 1

Example:

```
{
  "id" : 2,
  "tzShift" : 0,
  "utcTime" : 1437987103,
  "upTime" : 9,
  "event" : "OutputChanged",
    "params" : {
    "port" : "led_secured",
    "state" : false
  }
}
```

SwitchStateChanged

Signals a switch state change (refer to the intercom configuration in Hardware | Switches).

Event parameters:

Parameter	Type	Description
switch	uint32	Switch number 1...4
state	boolean	Current logic state of the switch: false – inactive, log.0 true – active, log.1
originator	string	Informs how the switch was activated. profile – by transition to the preset active time profile. api – by http api (api/switch/ctrl). ap – by user authentication at the access point. The event is then completed with app and session. rex – by pressing the exit button (that opens the door for a defined period of time for the person to leave the room). idt – by http api (api/switch/ctrl) if special authentication for 2N® Indoor Touch 2.0, 1.0 was used. dtmf – by dtmf code in the call. auth – authorization by a user / universal / zone code. uni – universal code authorization. zone – zone code authorization. automation – by an automation action.

Example:

```
{
  "id" : 2,
  "tzShift" : 0,
  "utcTime" : 1437987103,
  "upTime" : 9,
  "event" : "SwitchStateChanged",
    "params" : {
        "switch" : 1,
        "state" : true
  }
}
```

CallStateChanged

Signals a setup/end/change of the active call state.

Event parameters:

Parameter	Type	Description
direction	string	Call direction: **incoming** – incoming call **outgoing** – outgoing call
state	string	Current call state: **connecting** – call setup in progress (outgoing calls only) **ringing** – ringing **connected** – call connected **terminated** – call terminated
peer	string	SIP URI of the calling (incoming calls) or called (outgoing calls) subscriber
session	uint32	Unique call identifier. Can also be used in the **/api/call/answer**, **/api/call/hangup** and **/api/call/status** functions.
call	uint32	TBD

Example:


```
 {
  "id" : 5,
  "tzShift" : 0,
  "utcTime" : 1438064126,
  "upTime" : 660,
  "event" : "CallStateChanged",
  "params" : {
    "direction" : "incoming",
    "state" : "ringing",
    "peer" : "sip:2229@10.0.97.150:5062;user=phone",
    "session" : 1,
    "call" : 1
  }
}
```

RegistrationStateChanged

Signals a change of the SIP account registration state.

Event parameters:

Parameter	Type	Description
sipAccount	uint32	SIP account number showing a state change: **1** – SIP account 1 **2** – SIP account 2
state	string	New SIP account registration state: **registered** – account successfully registered **unregistered** – account unregistered **registering** – registration in progress **unregistering** – unregistration in progress

Example:

```
{
  "id" : 3,
  "tzShift" : 0,
  "utcTime" : 1437987105,
  "upTime" : 11,
  "event" : "RegistrationStateChanged",
  "params" : {
    "sipAccount" : 1,
    "state" : "registered"
  }
}
```

TamperSwitchActivated

Signals tamper switch activation - device cover opening. Make sure that the tamper switch function is configured in the Digital Inputs | Tamper Switch menu.

Event parameters:

Parameter	Type	Description
state	string	Tamper switch state: **in** – signals tamper switch activation (i.e. device cover open). **out** – signals tamper switch deactivation (device cover closed).

Example:

```
{
  "id" : 54,
  "tzShift" : 0,
  "utcTime" : 1441357589,
  "upTime" : 158,
  "event" : "TamperSwitchActivated",
  "params" : {
    "state" : "in"
  }
}
```

UnauthorizedDoorOpen

Signals unauthorized door opening. Make sure that a door-open switch is connected to one of the digital inputs and the function is configured in the Digital Inputs | Door State menu.

Event parameters:

Parameter	Type	Description
state	string	Unauthorized door opening state: in – signals the beginning of the unauthorized opening state. out – signals the end of the unauthorized door opening state.

Example:

```
{
  "id" : 80,
  "tzShift" : 0,
  "utcTime" : 1441367842,
  "upTime" : 231,
  "event" : "UnauthorizedDoorOpen",
  "params" : {
    "state" : "in"
  }
}
```

DoorOpenTooLong

Signals an excessively long door opening or failure to close the door within a timeout. Make sure that a door-open switch is connected to one of the digital inputs and the function is configured in the Digital Inputs | Door State menu.

Event parameters:

Parameter	Type	Description
state	string	DoorOpenToo Long state: in – signals the beginning of the DoorOpenTooLong state. out – signals the end of the DoorOpenTooLong state.

Example:

```
{
  "id" : 96,
  "tzShift" : 0,
  "utcTime" : 1441369745,
  "upTime" : 275,
  "event" : "DoorOpenTooLong",
  "params" : {
    "state" : "out"
  }
}
```

LoginBlocked

Signals a temporary blocking of the web interface access due to repeated entering of an invalid login name or password.

Event parameters:

Parameter	Type	Description
address	string	IP address from which invalid data were entered repeatedly.

Example:

```
{
  "id" : 5,
  "tzShift" : 0,
  "utcTime" : 1441369745,
  "upTime" : 275,
  "event" : "LoginBlocked",
  "params" : {
    "address" : "10.0.23.32"
  }
}
```

UserAuthenticated

Signals user authentication and subsequent door opening.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit
session	string	Informs how many times the user has been authenticated.
name	string	Specifies the name of the phone book user.
uuid	string	User's unique ID
apbBroken	string	Tapped card validity in Anti-passback. **false** – inactive soft APB **true** – active and broken soft ABP

Example:


```
{
  "success" : true,
  "result" : {
    "events" : [
      {
        "id" : 65,
        "tzShift" : 0,
        "utcTime" : 1593606655,
        "upTime" : 7951,
        "event" : "UserAuthenticated",
        "params" : {
          "ap" : 0,
          "session" : 6,
          "name" : "Alice Gruberov\u00E1",
          "uuid" : "8fa29ebc-2fe8-4a8c-9a3b-d8b0351fb6f8",
          "apbBroken" : true
        }
      }
    ]
  }
}
```

CardHeld

Signals that an RFID card has been tapped on the reader for more than 4 s.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit
session	string	Informs how many times the card has been applied.
direction	string	RFID direction: in – arrival out – departure any – passage *Note: Set the card reader direction using the intercom configuration interface.*
reader	string	Identification of the reader that read the card.
uid	string	User uid for devices connected to the Access Commander only. Devices disconnected from the Access Commander do not send this parameter.
valid	string	true, false

Example:


```
{
  "id" : 9,
  "tzShift" : 0,
  "utcTime" : 1516893493,
  "upTime" : 354,
  "event" : "CardHeld",
  "params" : {
    "ap" : 1,
    "session" : 4,
    "direction" : "out",
    "reader" : "ext2",
    "uid" : "3F00F318E7",
    "valid" : true
   }
}
```

SilentAlarm

Signals silent alarm activation.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
session	string	Informs how many silent alarms have been activated.
name	string	Specifies the phonebook username.

Example:


```
{
  "id" : 200,
  "tzShift" : 0,
  "utcTime" : 1548079445,
  "upTime" : 2053,
  "event" : "SilentAlarm",
  "params" : {
    "ap" : 0,
    "session" : 17,
    "name" : "Joseph",
    "uuid" : "54877b0e-4cc3-c645-9530-6c7850f47a9c"
  }
}
```

AccessLimited

Signals rejection of the set user.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
type	string	card, code, finger
state	string	State, available values: in = active, out = inactive.

Example:


```
{
    "id" : 408,
    "tzShift" : 0,
    "utcTime" : 1517302112,
    "upTime" : 408951,
    "event" : "AccessLimited",
    "params" : {
      "ap" : 0,
      "type" : "card",
      "state" : "in"
    }
}
```

PairingStateChanged

Signals pairing with a Bluetooth interface.

Event parameters:

Parameter	Type	Description
state	string	pending
authId	string	Authorisation ID

Example:

```
{
  "id" : 197,
  "tzShift" : 0,
  "utcTime" : 1516894499,
  "upTime" : 1360,
  "event" : "PairingStateChanged",
  "params" : {
    "state" : "pending",
    "authId" : "F2CAE955C9B4E81CD00E3A096E52543B"
  }
}
```

SwitchesBlocked

Signals lock blocking by the tamper switch. If the function is enabled, all the switches get blocked for 30 minutes whenever the tamper is activated. Blocking is active even after the device restart

Event parameters:

Parameter	Type	Description
state	string	in, out

Example:

```
{
  "id" : 205,
  "tzShift" : 0, "utcTime" : 1516894667,
  "upTime" : 1528,
  "event" : "SwitchesBlocked",
  "params" : {
    "state" : "in"
  }
}
```

FingerEntered

Signals that a finger has been tapped on the biometric reader.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
session	string	Informs how many times the finger has been enrolled.
direction	string	Fingerprint reader passage direction: "in" – entry "out" – exit "any" – any direction *Note: Set the reader passage direction via the intercom configuration interface.*
uuid	string	User's unique ID
valid	string	Fingerprint validity (if available as a valid user fingerprint in the configuration) false – invalid fingerprint true – valid fingerprint

Example: Reading of a user's fingerprint


```
{
  "id" : 1368,
  "tzShift" : 0,
  "utcTime" : 1548145535,
  "upTime" : 62598,
  "event" : "FingerEntered",
  "params" : {
    "ap" : 0,
    "session" : 1,
    "direction" : "in",
    "valid" : false
  }
}
```


Unsuccessful specification: Reading of an unset user's fingerprint


```
{
    "id" : 14,
    "tzShift" : 0,
    "utcTime" : 1511859513,
    "upTime" : 65887,
    "event" : "FingerEntered",
    "params" : {
        "session" : 3,
        "valid" : false
    }
}
```

MobKeyEntered

Signals Bluetooth reader authentication.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
session	string	Informs how many times the Mobile KEY authorisation has been applied.
direction	string	Passage direction: "in" – entry "out" – exit "any" – any direction *Note: Set the reader passage direction via the intercom configuration interface.*
authid	string	Mobile Key ID.
uuid	string	User's unique ID
valid	string	Mobile Key validity (if available as a valid user Mobile Key in the configuration) false – invalid Mobile Key true – valid Mobile Key

Example:


```
{
  "id" : 161,
  "tzShift" : 0,
  "utcTime" : 1548079174,
  "upTime" : 1782,
  "event" : "MobKeyEntered",
  "params" : {
    "ap" : 0,
    "session" : 9,
    "direction" : "in",
    "authid" : "48c48155eed7ea1dbb0b4d534b7459b9",
    "uuid" : "54877b0e-4cc3-c645-9530-6c7850f47a9c",
    "valid" : true
  }
}
```

DoorStateChanged

Signals a door state change.

Event parameters:

Parameter	Type	Description
state	string	opened, closed

Example:

```
{
        "id" : 240,
        "tzShift" : 0,
        "utcTime" : 1516895295,
        "upTime" : 2156,
        "event" : "DoorStateChanged",
        "params" : {
          "state" : "opened"
        }
}
```

UserRejected

Signals user authentication rejection.

Event parameters:

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
session	string	Informs how many times the authorisation has been rejected.
name	string	User name
uuid	string	User's unique ID
reason	string	apLocked, invalidTime, invalidProfile, invalidSequence, invalidCredential, authInterrupted, timeout, switchDisabled

Example:


```
{
  "id" : 173,
  "tzShift" : 0,
  "utcTime" : 1548079274,
  "upTime" : 1882,
  "event" : "UserRejected",
  "params" : {
    "ap" : 0,
    "session" : 10,
    "name" : "Joseph",
    "uuid" : "54877b0e-4cc3-c645-9530-6c7850f47a9c",
    "reason" : "invalidCredential"
  }
}
```

DisplayTouched

Signals display touch.

Event parameters:

Parameter	Type	Description
x	string	Display touch point coordinate. The maximum value depends of the display resolution.
y	string	Display touch point coordinate.
dx	string	Coordinate change due to movement on the display; negative values are possible. The maximum value depends of the display resolution.
dy	string	Coordinate change due to movement on the display.

Example:


```
{
    "id" : 337,
    "tzShift" : 0,
    "utcTime" : 1517301424,
    "upTime" : 408263,
    "event" : "DisplayTouched",
    "params" : {
      "x" : 89,
       "y" : 100,
      "dx" : 0,
      "dy" : 0
    }
}
```

DtmfEntered

Signals a DTMF code in the call.


```
{
    "id" : 86,
    "tzShift" : 0,
    "utcTime" : 1558522871,
    "upTime" : 3531,
    "event" : "DtmfEntered",
    "params" : {
      "code" : "00",
      "type" : "uni",
      "call" : 3,
      "valid" : true
 }
}
```


Parameter	Type	Description
code	string	Display the code characters entered.
type	string	The code type used. **uni** – universal switch code **user** – user code
call	string	Call ID.
valid	string	Code validity (i.e. the valid universal switch code or valid user code). **false** – invalid code **true** – valid code

AccessTaken

Signals that a card has been tapped in the Anti-passback area.

```
{
  "success" : true,
  "result" : {
    "events" : [

    ]
  }
}
```

ApLockStateChanged

Signals an emergency lockdown state change (on/off).

```
{
    "id" : 35,
    "tzShift" : 0,
    "utcTime" : 1558522465,
    "upTime" : 3125,
    "event" : "ApLockStateChanged",
    "params" : {
      "ap" : 0,
      "state" : "in"
    }
}
```

Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
state	string	Status change state. **"in"** – beginning of the emergency lockdown interval **"out"** – end of the emergency lockdown interval

RexActivated

Signals the input activation set for the REX button.


```
{
    "id" : 29,
    "tzShift" : 0,
    "utcTime" : 1558522162,
    "upTime" : 2822,
    "event" : "RexActivated",
    "params" : {
      "ap" : 1,
      "session" : 1
    }
}
```


Parameter	Type	Description
ap	string	Access Point, available states: 0 = entry, 1 = exit.
session	string	Display how many times the REX button has been activated.

LiftStatusChanged

Signals the Lift Control module connection/disconection.


```
  {
        "id" : 2871,
        "tzShift" : 0,
        "utcTime" : 1561540370,
        "upTime" : 73822,
        "event" : "LiftStatusChanged",
        "params" : {
          "module" : 0,
          "ready" : true
        }

},
```


Parameter	Type	Description
module	string	Display the ID module.
ready	string	

LiftFloorsEnabled

Signals permanent access to a floor or permanent user access.

```
{
        "id" : 2850,
        "tzShift" : 0,
        "utcTime" : 1561540011,
        "upTime" : 73463,
        "event" : "LiftFloorsEnabled",
        "params" : {
          "type" : "user"
          "floors" : [
            0, 1, 2, 3, 4
          ],
          "uuid" : "621a5a49-1f8b-d34c-9a8b-881055864deb",
        }
      },
```

```
{
        "id" : 2855,
        "tzShift" : 0,
        "utcTime" : 1561540016,
        "upTime" : 73468,
        "event" : "LiftFloorsEnabled",
        "params" : {
          "type" : "public"
          "floors" : [
            1, 4
          ],
        }
      },
```

Parameter	Type	Description
type	string	Provides information on the access type. **public** – change of public access **user** – user authentication
floors	string	Provides information on the accessible floors.

LifConfigChanged

Signals a change in the lift control configuration.


```
{
        "id" : 2860,
        "tzShift" : 0,
        "utcTime" : 1561540163,
        "upTime" : 73615,
        "event" : "LiftConfigChanged",
        "params" : {
          "hash" : 11
        }
      },
```


Parametr	Typ	Popis
hash	string	Unique configuration code.

CapabilitiesChanged

Signals a change in available functions.


```
{
  "success":true,
  "result":{
   "events":[
     {
      "id":21,
      "tzShift":0,
      "utcTime":1585037151,
      "upTime":256,
      "event":"CapabilitiesChanged",
      "params":{

      }

     }


   ]

  }

}
```

Parameter	Type	Description
id	string	Event sequence number.
tzShift	uint32	Difference between the local time and UTC in minutes. Add this value to utcTime to get the local time of event generation according to the time zone setting in the device: localTime = utcTime + tzShift * 60
utcTime	uint32	Absolute event generation time (Unix Time, UTC – Coordinated Universal Time).
upTime	uint32	Relative event generation time (seconds after the intercom restart).
event	string	CapabilitiesChanged event type.
params	object	Specific parameters of the event.

5.11 api audio

The following subsections detail the HTTP functions available for the **api/audio** service.

- 5.11.1 api audio test

5.11.1 api audio test

The **/api/audio/test** function launches an automatic test of the intecom built-in microphone and speaker. The test result is logged as an **AudioLoopTest** event.

The function is part of the **Audio** service and the user must be assigned the **Audio Control** privilege for authetication if required. The function is only available with the Enhanced Integration and Enhanced Audio licence key.

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/audio/test
{
  "success" : true
}
```

5.12 api email

The following subsections detail the HTTP functions available for the **api/email** service.

- 5.12.1 api email send

5.12.1 api email send

The **/api/email/send** function sends an e-mail to the required address. Make sure that the SMTP service is configured correctly for the device (i.e. correct SMTP server address, login data etc.).

The function is part of the **Email** service and the user must be assigned the **Email Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only.

The **GET** or **POST** method can be used for this function.

Request parameters:

Parameter	Description
to	Mandatory parameter specifying the delivery address.
subject	Mandatory parameter specifying the subject of the message.
body	Optional parameter specifying the contents of the message (including html marks if necessary). If not completed, the message will be delivered without any contents.
pictureCount	Optional parameter specifying the count of camera images to be enclosed. If not completed, no images are enclosed. Parameter values: [0, 5].
timeSpan	Optional parameter specifying the timespan in seconds of the snapshots enclosed to the email. Default value: 0.
width	Optional parameters specifying the resolution of camera images to be enclosed. The image height and width must comply with one of the supported options (see **api/camera/caps**).
height	

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/email/send?to=somebody@email.com&subject=Hello&body=Hello
{
  "success" : true
}
```

5.13 api pcap

The following subsections detail the HTTP functions available for the **api/pcap** service.

- 5.13.1 api pcap
- 5.13.2 api pcap restart
- 5.13.3 api pcap stop
- 5.13.4 api pcap live
- 5.13.5 api pcap live stop
- 5.13.6 api pcap live stats

5.13.1 api pcap

The **/api/pcap** function helps download the network interface traffic records (pcap file). You can also use the **/api/pcap/restart** a **/api/pcap/stop** functions for network traffic control.

The function is part of the **System** service and the user must be assigned the **System Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and the downloaded file can be opened directly in Wireshark, for example.

Example:

```
GET /api/pcap
```

5.13.2 api pcap restart

The **/api/pcap/restart** function deletes all records and restarts the network interface traffic recording.

The function is part of the **System** service and the user must be assigned the **System Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/pcap/restart
{
  "success" : true
}
```

5.13.3 api pcap stop

The **/api/pcap/stop** function stops the network interface traffic recording.

The function is part of the **System** service and the user must be assigned the **System Control** privilege for authentication if required. The function is available with the Enhanced Integration licence key only

The **GET** or **POST** method can be used for this function.

The function has no parameters.

The reply is in the **application/json** format and includes no parameters.

Example:

```
GET /api/pcap/restart
{
  "success" : true
}
```

5.13.4 api pcap live

The **api/pcap/live** function is used for starting of the chunked packet capture.
Service and Privileges Groups

- Service group is System.
- Privileges group is System Control.

Methods

- GET
- POST

Request

The request contains parameters in the URL (or in the **application/x-www-form-urlencoded** format when POST is used).

Table 1. Request Parameters

Parameter Name	Mandatory	Expected Values	Default Value	Description
duration	No	Integer defining duration of download in seconds	indefinitely	Defines the duration of the packet capture. If the parameter is omitted or 0, the duration is infinite (i.e. until it is stopped by **api/pcap/live/stop** or the download is severed by the target device).

Example of a Request

```
URL: https://192.168.1.1/api/pcap/live?duration=10
```

Response

The device starts streaming chunked data upon a successful request.

Example of Using Python to Download the Packet Capture

```
command = requests.post( "https://" + address + "/api/pcap/live?
duration=120", verify=False, stream=True, auth=HTTPBasicAuth("admin", "p
ass") ) with open("trace.pcap", 'wb') as f: for chunk in command.
iter_content(chunk_size=None): f.write(chunk)
```

If a packet capture is already running, another packet capture cannot be started.

5.13.5 api pcap live stop

The **api/pcap/live/stop** function is used for stopping of the chunked packet capture.

Service and Privileges Groups

- Service group is System
- Privileges group is System Control

Methods

- GET
- POST

Request

The request does not have any parameters.

Example of a Request

```
URL: https://192.168.1.1/api/pcap/live/stop
```

Response

The device stops streaming chunked data upon a successful request. The request can help stop a capture without a set duration or stop a capture with a duration value prematurely.

The device replies with **success** : **true** even if there is no running capture. There are no specific error codes for this endpoint.

5.13.6 api pcap live stats

The **api/pcap/live/stats** function is used for getting of status of the chunked packet capture.

Service and Privileges Groups

- Service group is System.
- Privileges group is System Control.

Methods

- GET
- POST

Request

The request does not have any parameters.

Example of a Request

```
URL: https://192.168.1.1/api/pcap/live/stats
```

Response

The response is in the **application/json** format. The response contains the **success** and **result** keys. The **result** value contains detailed information on the packet capture status.

Table 1. Response JSON Keys

Key	Typical Returned Values	Description
running	`true` or `false`	Indicates whether the chunked packet capture is currently running or not.
bytesSent	Integer	Contains the number of bytes sent in the currently running open packet capture. If the packet capture is not running, the value is 0.
packetsSent	Integer	Contains the number of packets sent in the currently running open packet capture. If the packet capture is not running, the value is 0.

Example of a Response

```
{ "success": true, "result": { "running": true, "bytesSent": 11261, "pac
ketsSent": 90 } }
```

5.14 api dir

The following subsections detail the HTTP functions available for the **api/dir** service.

- 5.14.1 api dir template
- 5.14.2 api dir create
- 5.14.3 api dir update
- 5.14.4 api dir delete
- 5.14.5 api dir get
- 5.14.6 api dir query

5.14.1 api dir template

The **/api/dir/template** function retrieves the template of an entry in the directory.

Methods

- GET
- POST

Request

Table 1. Request Parameters

Key Name	Mandatory	Expected Values	Default Value	Description
N/A	-	-	-	-

Example of a Request

```
https://192.168.1.1/api/dir/template
```

Response

The response is in the **application/json** format. The **result** object contains the keys **series** and **users**.

Go to the topic **api/dir/query** to get more information on the use of the key **series**.

The key **users** contains an array with one object (entry template) that contains all available keys of an entry in the directory with their default values for a particular device.

Tip

- You can get better acquainted with the structure of the JSON response in the example at the end of this topic.

Note

- Available keys depend on the model, type and hardware configuration of a device (e.g. key photo is only applicable for devices that have a display and store images in their directories).

Table 2. Response JSON Keys in the **users** Array

Key	Typical Returned Values	Description
uuid	Empty	Unique User Identifier. When a new entry in the directory is created by **api/dir/create**, uuid can be either provided as a parameter of the request or automatically generated by the device. The format of uuid is "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", where X can be any hexadecimal digit. All zeroes are reserved for an empty uuid.
deleted	`false`	Indication of whether an entry in the directory has been deleted or not. Deleted entries remain in the directory until the maximum number of entries is reached. Stored deleted entries preserve uuid only for logging reasons. Two valid values: `false`, `true`.
owner	Empty	Indication of whether an entry in the directory is remotely managed by 2N® My2N, 2N® Access Commander or another remote management system. This value is intended for internal 2N® TELEKOMUNIKACE a.s. usage, alternatively it can be used for deleting a group of users (see **api/dir/delete**).
name	Empty	Name of an entry in the directory (a user or device name). A string of up to 63 characters is expected. The name can be left empty (the entry is in such a case identified by its uuid in logs, emails, etc.).
photo	Empty	Image of an entry in the directory (e.g. user's photo, company logo). Saved as base64 encoded jpeg with the following syntax: EXAMPLE: *_DATA_IN_BASE64*
email	Empty	Email address of an entry in the directory. The expected format is *namestructure@domainhierarchy.top* . It is possible to enter multiple addresses separated with commas (saved as one string).

treepath	/	Definition of positions of an entry in the directory on the display. • The default position is in the root folder. This position is achieved by simply entering only one slash. **EXAMPLE:** */* shows the entry in a root folder • An entry may be positioned on a display several times - the positions are separated with a semi-colon (**;**). **EXAMPLE:** */Folder1/;/Folder2/* shows the entry both in Folder1 and Folder2 • An entry may be assigned a calling group that also serves as an alternative name in an individual position on the display by omitting the slash at the end of the position definition. **EXAMPLE:** */Folder1/Calling Group* shows the entry in Folder1 under the name "Calling Group" • An entry may be prioritized (i.e. a prioritized entry is shown above unprioritized entries) individually in each position by adding **:1** at the end of the position definition. **EXAMPLE:** */Folder1/:1;Folder2/Calling Group:1* shows the entry prioritized in Folder1 under its name and the entry prioritized in Folder2 as "Calling Group". • Multiple entries may be assigned to one calling group (selecting this calling group on a display results in a group call to all the entries under the calling group). **EXAMPLE:** *Entry1:* */Calling Group* *Entry2:* */Calling Group* shows both entries in the root folder as one row named "Calling Group" (both entries are called when this row is selected)
virtNumber	Empty	Virtual number of an entry in the directory. Virtual numbers may be dialed using a dialpad (if configured). The maximum length is 7 characters. The first and the last character may be chosen from the range of A to Z or 0 to 9. The rest of the characters may be between 0 to 9.
deputy	Empty	Uuid of a deputy entry that is called when the original callee is unavailable or not answering. When the deputy is not set the deputy uuid, it remains empty.
buttons	Empty	

Key	Typical Returned Values	Description
		Buttons assigned to this entry in the directory. An array of integers (assigned according to the button position starting from 1) separated by a comma.

callPos	Array	Calling information of an entry. Entered as an array of up to three objects, i.e. up to three sets of calling information can be entered. Each of these three objects can contain the following keys:
		• peer - a phone number of an entry in the directory
		• profiles - time profiles if this phone number is valid (a number is not dialed when invalid)
		  • P=X,..,Y
		  where X,..,Y stands for a comma-separated array of predefined time profiles from 0 (time profile 1) to 19 (time profile 20)
		  EXAMPLE: *P=1,3,5* means that the predefined profiles 2, 4 and 6 define the validity period of the phone number
		  • D=A,..,B@H:MM-H:MM
		  where A,..,B stands for a comma-separated array of days of week (0 to 6 from Sunday to Saturday, 7 is Holiday), H stands for hour of the day (from 0 to 23), MM stands for minutes (from 00 to 59) - two values defining an interval. Several definitions may be combined separated with a semi-colon.
		  EXAMPLE: *D=7@0:15-13:15;D=5,7@13:15-15:15;D=7@15:15-23:30* means that the phone number is valid on Holiday from 0:15 to 13:15, on Friday and Holiday from 13:15 to 15:15 and on Holiday from 15:15 to 23:30.
		  EXAMPLE: *D=5,7* means that the phone number is valid on Friday and Holiday for the whole day.
		• grouped - defines whether the number is dialed in a group call together with the previous number (the third number is dialed with a deputy). Can be `true` or `false`.
		• ipEye - defines the IP address of the PC on which the 2N® IP Eye application is running (used for receiving video if the telephone does not have a display).

access	JSON Object	Access Control information of an entry in the directory. Contains the following keys:

Key	Typical Returned Values	Description

		• validFrom - definition of the start of the validity term for the credentials of an entry in the directory. Entered in the Unix Time format. If left empty, the validity period starts at the beginning of the time (i.e. 00:00:00 UTC Jan 1st 1970).
		• validTo - definition of the end of the validity term for the credentials of an entry in the directory. Entered in the Unix Time format. If left empty, the validity period ends at the end of the time (i.e. 03:14:07 UTC Jan 19th 2038).
		• accessPoints - contains an array of access points (two access points, entry and exit). Each array item is represented as a JSON object with the following keys:
		  • enabled - defines whether it is generally possible to use this access point (i.e. if it is possible to authenticate at that particular access point). Two valid values: `true`, `false`.
		  • profiles - defines whether it is currently possible to use this access point (i.e. if it is possible at the moment to authenticate at that particular access point). The syntax of the profile definition is the same as in callPos.
		• pairingExpired - defines whether the Bluetooth pairing of an entry in the directory has expired or not. Two valid values: `true`, `false`.
		• virtCard - defines the virtual card of an entry in the directory that is relayed to Wiegand and other 3rd party systems if there is a successful authentication of the entry in the directory. 6 to 32 hexadecimal characters are expected.
		• card - defines up to two cards of an entry in the directory that are used for authentication. The card numbers are written as an array of strings. 6 to 32 hexadecimal characters are expected.
		• mobkey - defines the Bluetooth authentication ID of an entry in the directory. 32 hexadecimal characters are expected.
		• fpt - fingerprint templates of an entry in the directory. An encoded fingerprint is expected (for the details contact the Technical Support staff).
		• pin - defines the pin of an entry in the directory. 2 to 15 digits are expected.
		• apbException - defines whether an entry in the directory has an exception from the anti-passback policy (e.g. if so, its credentials can be used multiple times for entry without any exit). Two valid values: `true`, `false`.

Key	Typical Returned Values	Description
		• code - defines up to three individual codes for switches. The individual switch codes are written in an array of strings (2 to 15 digits). The first position in the array defines an individual code for the first switch. Input an empty string to skip a code for the particular switch. • liftFloors - defines the configuration of accessible lift floors upon authentication. The following formatting is expected: F=P,..,R@PROFILE1_DEFINITION|F=X,..,Z@PROFILE2_DEFINITION where P,..,R and X,..,Z are arrays of comma-separated floors (0 to 63). io_1_1 is entered as 0, io_1_5 is entered as 4, io_2_2 is entered as 9 and so on. The arrays of floors active in certain profiles are separated by |. Predefined profiles or ad hoc definition of a new profile can be used (see the syntax definition in callPos/profiles). The profile definition part can be omitted if no profile is applied. *EXAMPLE: F=2,4* defines floors without any time profile (user can access them any time). *EXAMPLE: F=5@P=0,7* defines that the sixth floor (F=5) is accessible the whole day on Mondays and Holidays. *EXAMPLE: F=0@P=7,13|F=0@D=5@9:15-11:45;D=4@11:45-13:30* defines that the first floor (F=0) is accessible in predefined profiles 8 and 14 (P=7,13) and in the time profile defined by the definition string. • licensePlates - defines a set of license plates configured to the user (used for opening upon license plate recognition). Individual licensePlates are separated by a comma. Whitespaces are ignored. A maximum of 255 characters is allowed (including whitespaces). The array is limited to 20 license plates and each license plate may have a maximum of 10 non-whitespace characters.
timestamp	0	A timestamp of performed changes in the directory. Timestamps are automatically generated by the directory in an ascending order. Go to the topic **api/dir/query** to get more information on the use of the timestamp.

Example of a Response


```
{
    "success": true,
    "result": {
        "series": "5247939846841727056",
        "users": [
            {
                "uuid": "",
                "deleted": false,
                "owner": "",
                "name": "",
                "photo": "",
                "email": "",
                "treepath": "\/",
                "virtNumber": "",
                "deputy": "",
                "buttons": "",
                "callPos": [
                    {
                        "peer": "",
                        "profiles": "",
                        "grouped": false,
                        "ipEye": ""
                    },
                    {
                        "peer": "",
                        "profiles": "",
                        "grouped": false,
                        "ipEye": ""
                    },
                    {
                        "peer": "",
                        "profiles": "",
                        "grouped": false,
                        "ipEye": ""
                    }
                ],
                "access": {
                    "validFrom": "0",
                    "validTo": "0",
                    "accessPoints": [
                        {
                            "enabled": true,
                            "profiles": ""
                        },
                        {
                            "enabled": true,
                            "profiles": ""
                        }
```

```
                    ],
                    "pairingExpired": false,
                    "virtCard": "",
                    "card": [
                        "",
                        ""
                    ],
                    "mobkey": "",
                    "fpt": "",
                    "pin": "",
                    "apbException": false,
                    "code": [
                        "",
                        "",
                        "",
                        ""
                    ],
                    "licensePlates": "",
                    "liftFloors": ""
                },
                "timestamp": 0
            }
        ]
    }
}
```


5.14.2 api dir create

The **/api/dir/create** function creates (or overwrites) an array of entries in the directory and sets their selected fields.

Methods

- PUT

Request

The request contains parameters in the **application/json** format. Go to the topic **api/dir/template** to get more information on various parameters of an entry in the directory and their representation.

Table 1. Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
force	No	`true`, `false`	`false`	This key selects whether the entry in the directory identified by uuid (see below) is overwritten by the new set of data.

Key Name	Mandatory	Expected Values	Default Value	Description
				When the value for this key is set to `true`, a new dataset overwrites the existing data and the remaining fields are reset to their default values. When it is set to `false` or omitted and there is an existing entry in the directory with the specified uuid, the device replies with error code `EDIR_UUID_ALREADY_EXISTS` and changes to the configuration are not processed. This key does not affect a creation of a new entry in the directory in any way.
users	No	array of JSON objects defining parameters of an entry in the directory	-	If this key is omitted or if its value is an empty array, the device response contains only the key series (no new entries in the directory are created). It is possible to submit empty objects in the array. The device creates an empty entry in the directory for each empty object (it is only assigned a uuid). If an object in the array contains the key uuid, an entry with the specified uuid is created or modified, or the device replies with an error code. If an object in the array does not contain uuid, the device creates a new entry and assign it a new uuid. The entry parameters are set to the values according to the keys defined in the JSON structure of a request. Study the example below. Go to the topic **api/dir/template** to get an overview of all available keys in the JSON definition of an entry in the directory.

Example of Request

```
URL: https://192.168.1.1/api/dir/create JSON { "force": true, "users":
[ { "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD",
"email": "abcd@def.cz", "access": { "pin": "1234" } }, { "name":
"ABCD2", "owner": "My2N", "email": "abcd2@def.cz" }, { "uuid":
"01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD3", "email":
"something", "access": { "pin": "5678" }, "test": "something",
"albert": "einstein" }, {}, {} ] }
```

If there is no entry in the directory with uuid 01234567-89AB-CDEF-0123-456789ABCDEF, the device creates an entry in the directory with this uuid and set its parameters name, email and access to the specified values.

If there is an entry in the directory with uuid 01234567-89AB-CDEF-0123-456789ABCDEF, the device overwrites its parameters name, email and access to the specified values and sets all of its other parameters to their default values (because the key force is set to true).

The device creates a second entry, assigns it a random uuid, sets its name, owner and email to the specified values and leaves the rest of its parameters at default values.

The third entry does not overwrite the existing entry with the same uuid because there are several errors (wrong email format, two non-existent fields referenced by **test** and **albert**).

Furthermore, two new empty entries are created (because there are two empty objects in the array). Each is assigned a random uuid, the rest of their parameters are set to default values.

Response

The response is in the **application/json** format. The **result** object contains the keys **series** and **users**.

Go to the topic **api/dir/query** to get more information on the use of the key **series**.

The key **users** contains an array of objects that contain keys and values of the result of the request (see the table below).

Tip

- You can get better acquainted with the structure of the JSON response in the example at the end of this topic.

Table 2. Response JSON Keys in the **users** Array

Key	Typical Returned Values	Description
uuid	uuid	Unique User Identifier of a created, modified or unchanged entry.
timestamp	integer	A timestamp of performed changes in the directory. Go to the topic **api /dir/query** to get more information on the use of the timestamp. Timestamp is present only when a change in the directory was successful.
errors	array of error objects	An error object containing an array of all errors that occured. **errors** object is present only when a change in the directory failed. It contains an error code in the value of the key **code** showing a reason for the failure of a change in the directory. It may contain a further specification of the error reason in the value of the key **field** for error codes `EDIR_FIELD_NAME_UNKNOWN` and `EDIR_FIELD_VALUE_ERROR` showing which key or value violates the validation rules. The following error codes may be returned in a response: • `EDIR_UUID_INVALID_FORMAT` - uuid is not in the valid format which is "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", where X can be any hexadecimal digit. All zeroes are reserved as an empty uuid. • `EDIR_UUID_ALREADY_EXISTS` - an entry with the specified uuid exists in the directory and the key **force** is set to false or omitted. Therefore the requested modification cannot be performed. • `EDIR_FIELD_NAME_UNKNOWN` - unknown key. The list of available keys for a particular device can be obtained using the **/api/dir /template** function. • `EDIR_FIELD_VALUE_ERROR` - the specified value does not fit validation criteria (the value is not valid). • `EDIRLIM_USER` - the directory is full. • `EDIRLIM_PHOTO` - the limit of photo storage has been reached. New entries can be created without photos. • `EDIRLIM_FPT` - the limit of fingerprint templates storage has been reached. New entries can be created without fingerprint templates. • `EINCONSISTENT` - there is an inconsistency in the values of the keys **validFrom** and **validTo** (**validFrom** has to be lower than **validTo**).

Example of Response


```
{ "success": true, "result": { "series": "64234076876064319 51", "users":
[ { "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "timestamp": 34 },
{ "uuid": "044197A7-54AD-7577-6EEA-787A6097263E", "timestamp": 35 }, { "
errors": [ { "code": "EDIR_FIELD_VALUE_ERROR", "field": "email" }, { "co
de": "EDIR_FIELD_NAME_UNKNOWN", "field": "test" }, { "code": "EDIR_FIELD
_NAME_UNKNOWN", "field": "albert" } ] }, { "uuid": "41970B83-21C8-45DD-
8FFC-787A6097263E", "timestamp": 36 }, { "uuid": "0447BBA7-6E7c-420C-
A654-466D43D6A067", "timestamp": 37 } ] } }
```


The first entry is created with the specified uuid and fields (unspecified fields are set to their default values). The entry gets created regardless whether there is an already existing entry with the same uuid because the key **force** in the request is set to `true`. The timestamp of the change is returned.

The second entry is created, assigned a random uuid and its specified fields are filled (unspecified fields are set to their default values). The timestamp of the change is returned.

The third object in the request contained an invalid email address format. Furthermore, non-existent fields were referenced by the keys **test** and **albert**.

The fourth and fifth entries where created successfully with randomly assigned uuids and all fields set to default values. The timestamp in the device updated twice to reflect that. The timestamps of the changes are returned.

Tip

- If the key **force** in the request is not set to `true`, any attempts to create an entry with the existing uuid end with error code `EDIR_UUID_ALREADY_EXISTS`.

5.14.3 api dir update

The **/api/dir/update** function updates an array of entries in the directory and sets their selected fields.

Methods

- PUT

Request

The request contains parameters in the **application/json** format. Go to the topic **api/dir/template** to get more information on various parameters of an entry in the directory and their representation.

Table 1. Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
users	Yes	array of JSON objects defining parameters of an entry in the directory	-	The array has to contain at least one object with the key **uuid**, which defines the entry to be updated. Go to the topic **api/dir/template** to get an overview of all available keys in the JSON definition of an entry in the directory.

Example of Request

```
URL: https://192.168.1.1/api/dir/update JSON { "users": [ { "uuid":
"01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD", "email":
"abcd@def.cz", "access": { "pin": "1234" } }, { "uuid": "76543210-68FF-
18CA-3210-FEDCBA987654", "name": "ABCD2", "owner": "My2N", "email":
"abcd2@def.cz" }, { "uuid": "01234567-89A-CDEF-0123-456789ABCDEF",
"name": "ABCD3", "owner": "My2N", "email": "abcd3@def.cz" }, { "uuid":
"01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD4", "owner":
"My2N", "email": "abcd4@def.cz", "albert": "einstein" }, { "uuid":
"01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD4", "owner":
"My2N", "email": "abcd4@def.cz", "access.pin": "hello" } ] }
```

If there is no entry in the directory with uuid 01234567-89AB-CDEF-0123-456789ABCDEF, the device will respond with an error code (see below). Similarly for the second uuid 76543210-68FF-18CA-3210-FEDCBA987654.

If the entry with uuid 01234567-89AB-CDEF-0123-456789ABCDEF is present, it will update its parameters according to the specified values for various keys. Similarly for the second uuid 76543210-68FF-18CA-3210-FEDCBA987654.

The third entry will not be updated (uuid has a wrong format).

The fourth entry will not be updated (unknow field name).

The fifth entry will not be updated (wrong format of access pin).

Response

The response is in the **application/json** format. The **result** object contains the keys **series** and **users**.

Go to the topic **api/dir/query** to get more information on the use of the key **series**.

The key **users** contains an array of objects that contain keys and values of the result of the request (see the table below).

Tip

- You can get better acquainted with the structure of the JSON response in the example at the end of this topic.

Table 2. Response JSON Keys in the **users** Array

Key	Typical Returned Values	Description
uuid	uuid	Unique User Identifier of a modified or unchanged entry.
timestamp	integer	A timestamp of performed changes in the directory. Go to the topic **api /dir/query** to get more information on the use of the timestamp. The timestamp is present only when a change in the directory was successful.
errors	array of error objects	An error object containing array of all errors that occured. **errors** key is present only when a change in the directory failed. It contains an error code in the key **code** showing a reason for the failure of a change in the directory. It may contain a further specification of the error reason in the key **field** for error codes `EDIR_FIELD_NAME_UNKNOWN` and `EDIR_FIELD_VALUE_ERROR` showing which key or value violates the validation rules. The following error codes may be returned in a response: • `EDIR_UUID_DOES_NOT_EXIST` - entry with the given uuid does not exist (i.e. cannot be updated). • `EDIR_UUID_IS_MISSING` - the mandatory key **uuid** is missing. • `EDIR_UUID_INVALID_FORMAT` - uuid is not in the valid format which is "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", where X can be any hexadecimal digit. All zeroes are reserved as an empty uuid.

Key	Typical Returned Values	Description
		• `EDIR_FIELD_NAME_UNKNOWN` - unknown key. The list of available keys for a particular device can be obtained using the **/api/dir/template** function. • `EDIR_FIELD_VALUE_ERROR` - a specified value does not fit the validation criteria (the value is not valid). • `EDIRLIM_PHOTO` - the limit of the photo storage has been reached. Photos cannot be added. • `EDIRLIM_FPT` - the limit of the fingerprint template storage has been reached. Fingerprint templates cannot be added. • `EINCONSISTENT` - there is an inconsistency in the values of the keys **validFrom** and **validTo** (**validFrom** has to be lower than **validTo**).

Example of Response


```
{ "success": true, "result": { "series": "64234076876064319 51", "users":
[ { "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "timestamp": 39 },
{ "uuid": "76543210-68FF-18CA-3210-FEDCBA987654", "errors": [ { "code":
"EDIR_UUID_DOES_NOT_EXIST" } ] }, { "uuid": "01234567-89A-CDEF-0123-
456789ABCDEF", "errors": [ { "code": "EDIR_UUID_INVALID_FORMAT" } ] },
{ "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "errors": [ { "code":
"EDIR_FIELD_NAME_UNKNOWN", "field": "albert" } ] }, { "uuid": "01234567-
89AB-CDEF-0123-456789ABCDEF", "errors": [ { "code": "EDIR_FIELD_VALUE_ER
ROR", "field": "access.pin" } ] } ] } }
```


The first entry in the directory is updated successfully, its **uuid** and **timestamp** of the change are returned.

The second entry does not exist (no entry with such **uuid**).

The third object has a wrong format of **uuid**.

An unknown key **albert** has been passed in the fourth object.

An invalid value of PIN has been passed in the fifth object.

5.14.4 api dir delete

The **/api/dir/delete** function deletes an array of entries in the directory.

Methods

- PUT

Request

The request contains parameters in the **application/json** format.

Table 1. Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
owner	Yes if users is omitted	a string	-	All entries in the directory with the specified owner are deleted
users	Yes if owner is omitted	array of JSON objects defining uuids	-	The array has to contain at least one object with the key **uuid**, which defines the entry that is to be deleted.

Example of Request

```
URL: https://192.168.1.1/api/dir/delete JSON (owner specified) {
"owner": "My2N" } JSON (uuid specified) { "users": [ { "uuid":
"01234567-89AB-CDEF-0123-456789ABCDEF" }, { "uuid": "76543210-68FF-18CA-
3210-FEDCBA987654" }, { "uuid": "76543210-68FF-18-3210-FEDCBA987654" }
] }
```

If there is no entry in the directory with the specified owner, an empty array is returned.

If there is no entry in the directory with uuid 01234567-89AB-CDEF-0123-456789ABCDEF, the device will respond with an error code (see below). Similarly for the second uuid 76543210-68FF-18CA-3210-FEDCBA987654.

If the entry with uuid 01234567-89AB-CDEF-0123-456789ABCDEF is present, it will be deleted. Similarly for the second uuid 76543210-68FF-18CA-3210-FEDCBA987654.

The third uuid has a wrong format and an error is returned.

Response

The response is in the **application/json** format. The **result** object contains the keys **series** and **users**.

Go to the topic **api/dir/query** to get more information on the use of the key **series**.

The key **users** contains an array of objects that contain keys **uuid** and **timestamp**.

- You can get better acquainted with the structure of the JSON response in the example at the end of this topic.

Table 2. Response JSON Keys in the **users** Array

Key	Typical Returned Values	Description
uuid	uuid	Unique User Identifier of a deleted or unchanged entry.
timestamp	integer	A timestamp of performed changes in the directory. Go to the topic **api /dir/query** to get more information on the use of the timestamp. The timestamp is present only when a change in the directory was successful.
errors	array of error objects	An error object containing an array of all errors that occured. **errors** object is present only when a change in the directory failed. It contains an error code in the key **code** showing a reason for the failure of a change in the directory. The following error codes may be returned in a response: • `EDIR_UUID_DOES_NOT_EXIST` - entry with the given uuid does not exist (i.e. cannot be deleted). • `EDIR_UUID_INVALID_FORMAT` - uuid is not in the valid format, which is "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", where X can be any hexadecimal digit. All zeroes are reserved as an empty uuid.

Example of Response

```
{ "success": true, "result": { "series": "6423407687606431951", "users":
[ { "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "timestamp": 39 },
{ "uuid": "76543210-68FF-18CA-3210-FEDCBA987654", "errors": [ { "code":
"EDIR_UUID_DOES_NOT_EXIST" } ] }, { "uuid": "76543210-68FF-18-3210-
FEDCBA987654", "errors": [ { "code": "EDIR_UUID_INVALID_FORMAT" } ] } ]
} }
```

The first entry in the directory is deleted successfully, its **uuid** and **timestamp** of the change are returned.

The second entry does not exist (no entry with such **uuid**).

The third object has a wrong format of **uuid**.

5.14.5 api dir get

The **/api/dir/get** function retrieves an array of entries in the directory and their specified fields.

Methods

- POST

Request

The request contains parameters in the **application/json** format. Go to the topic **api/dir/template** to get more information on various parameters of an entry in the directory and their object representation.

Table 1. Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
fields	No	array of strings	all fields with non-default values	Specify the names of the required fields in the response, if the key is not specified, all fields with non-default values are returned, if an empty array is submitted, all available fields are returned. Go to the topic **api/dir/template** to get an overview of all available keys in the JSON definition of an entry in the directory. Unknown field names are ignored.

Key Name	Mandatory	Expected Values	Default Value	Description
users	No	array of JSON objects defining uuids	-	The array has to contain at least one object with the key **uuid**, which defines the entry whose fields are to be returned. If the key is missing or an empty array is submitted, an empty array is returned.

Example of Request

```
URL: https://192.168.1.1/api/dir/get JSON { "fields": [ "name",
"email", "callPos.peer", "callPos[1].grouped" ], "users": [ { "uuid":
"01234567-89AB-CDEF-0123-456789ABCDEF" }, { "uuid": "76543210-68FF-18CA-
3210-FEDCBA987654" }, { "uuid": "76543210-68FF-18-3210-FEDCBA987654" }
] }
```

If there is no entry in the directory with uuid 01234567-89AB-CDEF-0123-456789ABCDEF, the device will respond with an error code (see below). Similarly for the second uuid 76543210-68FF-18CA-3210-FEDCBA987654.

If the entry with uuid 01234567-89AB-CDEF-0123-456789ABCDEF is present, its specified fields (in the example name, email, phone numbers of all calling destinations and for the second calling destination also grouped) will be returned. Similarly for the second uuid 76543210-68FF-18CA-3210-FEDCBA987654.

The uuid 76543210-68FF-18-3210-FEDCBA987654 is in a wrong format.

Response

The response is in the **application/json** format. The **result** object contains the keys **series** and **users**.

Go to the topic **api/dir/query** to get more information on the use of the object **series**.

The key **users** contains array of objects that contain keys and values of the result of the request (see the table below).

Tip

- You can get better acquinted with the structure of the JSON response in the example at the end of this topic.

Table 2. Response JSON Keys in the **users** array

Key Name	Typical Returned Values	Description
uuid	uuid	Unique User Identifier of a found entry.
Various keys	various	Specified fields of an entry in the directory that are returned. See the **api /dir/template**.
timestamp	integer	A timestamp of the last performed changes for each returned entry in the directory. Go to the topic **api/dir/query** to get more information on the use of the timestamp. The timestamp is present only when an entry in the directory is returned.
errors	array of error objects	An error object containing an array of all errors that occured. **errors** object is present only when a change in the directory failed. It contains an error code in the key **code** showing a reason for the failure of a change in the directory. The following error codes may be returned in a response: • `EDIR_UUID_DOES_NOT_EXIST` - entry with the given uuid does not exist (i.e. cannot be updated). • `EDIR_UUID_INVALID_FORMAT` - uuid is not in the valid format which is "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", where X can be any hexadecimal digit. All zeroes are reserved as an empty uuid.

Example of Response


```
{ "success": true, "result": { "series": "6423407687606431951", "users":
[ { "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD", "ema
il": "abcd@abcd.cz", "callPos": [ { "peer": "" }, { "peer": "", "grouped
": "false" }, { "peer": "" } ], "timestamp": 39 }, { "uuid": "76543210-
68FF-18CA-3210-FEDCBA987654", "errors": [ { "code": "EDIR_UUID_DOES_NOT_
EXIST" } ] }, { "uuid": "76543210-68FF-18-3210-FEDCBA987654", "errors":
[ { "code": "EDIR_UUID_INVALID_FORMAT" } ] } ] } }
```


The first entry in the directory is returned successfully, its **uuid** and **timestamp** are returned.

The second entry does not exist (no entry with such **uuid**).

The third object has a wrong format of **uuid**.

5.14.6 api dir query

The **/api/dir/query** function retrieves an array of entries in the directory defined by timestamp iterator and their specified fields.

Methods

- POST

Request

The request contains parameters in the **application/json** format. Go to the topic **api/dir/template** to get more information on various parameters of an entry in the directory and their object representation.

Table 1. Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
series	No	string	current device series	The string represent a number of the timestamps series in the device. If the key is not submitted, the current device series is considered. If the specified series differs from the current device series, the device will return its series, the highest timestamp value and invalid timestamp.
fields	No	array of strings	all fields with non-default values	Specify the names of the required fields in the response, if the key is not specified, all fields with non-default values are returned, if an empty array is submitted, all available fields are returned. Go to the topic **api/dir/template** to get an overview of all available keys in the JSON definition of an entry in the directory. Unknown field names are ignored.
iterator	No	JSON object	{"timestamp": 0}	The key defines the iterator for the query (timestamp iterator is supported). The timestamp iterator has an integer value timestamp indicating the first directory entry to be returned (the last timestamp is always returned by any of **api/dir/create**, **api/dir**

Key Name	Mandatory	Expected Values	Default Value	Description
				/update or api/dir/delete). If the timestamp iterator value is zero, all directory entries are returned. If the timestamp iterator value is higher than the current timestamp value in the device, the device will return its series, the highest timestamp value and invalid timestamp.

Example of Request

```
URL: https://192.168.1.1/api/dir/query JSON { "series":
"2229480630597592840", "fields": [ "name", "email", "callPos.peer",
"callPos[1].grouped" ], "iterator": { "timestamp": 6 } }
```

If the series is inconsistent with the current series in the device, the device returns its current series, the maximum value of the timestamp and invalid timestamp.

If the specified timestamp is lower than the current maximum timestamp, all the higher timestamps are returned.

The device is capable of handling of up to 10000 unique user identifiers. Once the number of uuids gets higher, the device returns key **invalid**, which indicates that there is an unknown history of the directory (there were entries in the directory that were deleted and the device no longer stores them).

If the specified timestamp is lower than the invalid timestamp, the device returns its current series, the maximum value of the timestamp and invalid timestamp.

Response

The response is in the **application/json** format. The **result** object contains the keys **series** and **users**.

The key **users** contains array of objects that contain keys and values of the result of the request (see the table below).

Tip

- You can get better acquainted with the structure of the JSON response in the example at the end of this topic.

Table 2. Response JSON Keys in the **users** array

Key Name	Typical Returned Values	Description
uuid	uuid	Unique User Identifier of a found entry.
Various keys	various	Specified fields of an entry in the directory that are returned. See the **api /dir/template**.
timestamp	integer	A timestamp of the last performed changes for each returned entry in the directory. The timestamp is present only when an entry in the directory is returned.

Example of Response


```
{ "success": true, "result": { "series": "2229480630597592840", "users":
[ { "uuid": "01234567-89AB-CDEF-0123-456789ABCDEF", "name": "ABCD", "ema
il": "abcd@abcd.cz", "callPos": [ { "peer": "" }, { "peer": "", "grouped
": "false" }, { "peer": "" } ], "timestamp": 6 }, { "uuid": "A6543210-
68FF-18CA-3210-FEDCBA987654", "name": "DEFG", "email": "defgd@defg.cz",
"callPos": [ { "peer": "" }, { "peer": "", "grouped": "false" }, { "peer
": "" } ], "timestamp": 9 }, { "uuid": "044197A7-54AD-7577-6EEA-
787A6097263E", "name": "HIJK", "email": "hijk@hijk.cz", "callPos": [ { "
peer": "" }, { "peer": "", "grouped": "false" }, { "peer": "" } ], "time
stamp": 10 } ] } }
```


Three entries in the directory that have timestamp 6 and higher are returned (in this case the maximum timestamp in the directory is 10).

5.15 api mobilekey

The following subsections detail the HTTP functions available for the api/mobilekey service.

- 5.15.1 api mobilekey config

5.15.1 api mobilekey config

The api/mobile/key/config function is used for reading and writing of location IDs and encryption keys for Bluetooth Authentication.

Service and Privileges Groups

- Service group is System.
- Privileges group is System Access.

Methods

- **GET** – read location IDs and encryption keys
- **PUT** – write location IDs or encryption keys

Request

There are no parameters used for **GET** request.

The **PUT** request contains parameters in the *application/json* format.

Table1. PUT Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
location	No	String of maximum length of 127 characters	–	*location* defines the specific device location for the purpose of Bluetooth authentication. Any string that defines the location uniquely is accepted. The location is broadcasted by the 2N devices and serves for selecting relevant authentication parameters by the Bluetooth authentication device.
keys	No		–	

Key Name	Mandatory	Expected Values	Default Value	Description
		Array of objects containing encryption keys		***keys*** contains encryption keys that are used for secure communication between a 2N device and a device used for authentication via Bluetooth. The objects in the array have the following keys: • ***type*** – algorithm type, RSA is currently supported, this key is optional, • ***key*** – encryption key data (DER format encoded in Base64), use 1024 bit encryption keys, this key is mandatory, • ***ctime*** – creation time represented as Unix time 32 bit unsigned integer, this key is optional.

Example of PUT Request

```
URL: https://192.168.1.1/api/mobilekey/config JSON: { "location":
"LocationUniqueID", "keys": [ { "type": "rsa", "key":
"MIICXAIBAAKBgQCXmIX1U7wGFW3FiDdhq7BIktIc4lg7X2IMxLE83I75S3BRPL
/7LCAefnMUJL0uyyFdeMpRoOVhVs
/iPfnYPNf4AiQO4lIQh8tSKDeat5IfXSMy9zXMyHeBOBg19R+
/uShyJsnLoJoB5MJDowwkOuSMIskK+dA17+3E
/Y+ujhhpCQIDAQABAoGAfzHOVAUp4cDhFbgxH5Y6lun5uZqAhXCGiEgQngxBOhJ97uuV+V0Q
pgVa8S/SPAzbtd2/g7YIQB
/i1OVDWJfUbEiuBhr6ZHwk5jfCfF0KkmTQtEBd4bbCz+FWyoy19DUXdsLNMf8GW4eWhooX+N
Cqc2sfloO4Nz+SpXmqpsMIc/ECQQDk63xVnRqPCgG3fqpLVGWkQl9wmYAIUP8MrdOAfRYfSC
/LrjX55lCRj4mAnSzRQfNclSEz2mITkoaCcjn1lTmXAkEAqYdjMEhIrg4LpYzqZDOF6v6w
/sUcLkepktTBYCFFV+YgOrlPr7akR8XtED8X/6QHwWciphp/5OBoJ
/KRAWWGXwJAZ0YNe5o6pxk+mQed0AotKKOA5w15A0d3KMMqzaag2k
/4sAzR8QGEi4aT4+AEngsAvV3R8tCsumO6JxNdLnu51QJAO6abzB1jFXtztajDwMYwVOORO9
P3eoFUtYmPEVgjoi4jIQabd2R4oZiPNaw9sYHyCKdVlcS1Q7+CZqv
/QdKLWQJBAKeoGxqcpDHvMtzgcSj3lZz0Z8dWmgtTF7QO5boHhxtZ1SEo3MvlicVue8U1tV2
XjUR6r7YueuusM9GxhBqr5YI=", "ctime": 1608047606 }, { "type": "rsa",
"key":
"MIICXQIBAAKBgQCfyMHsTjPKf3Dv00gwMrQAR5UZrpt3tBy3kBvPv4R4o9H7Zzse7+yKwfP
TddKJQOL1IrCX06Zo8SZAMotjjpMy1M9K27ZB95YtAYiGLLRWeLAJUkL4gixgkHeS+T8uQxL
W7
/etqwU00uPmd94ZEZy226CHdKQW3zge2WEtuQ5oCwIDAQABAoGAZCp6RyUPGpahufZ9fpmKd
dJqCduH4paqmfhhNu8coHQyIqQoT9CgPKwxqhJmlVxz6rCAe+1WmNrz27LT5uluJKViU0XnL
V7FHG2smagjQ3rPepgOGcayphuiIlHikaBCafxnCRV
/E1Ifg08d1xK4cK858yMjpoEgDdEJi0R2qmECQQDXqtwGiXYSRnZzR90eCjrip6IIQqJuARE
91LOLyOhkPzCiPPf2IrT1JQsw6Tu0ZTm3NJzZ0VSEdZU6s2NcKHsnAkEAvap5GacBi9EZ9ls
iaQj
/dVA6LbUnBCo7qwRj7SUYw6ikCvmcLjdpjR80twj3FTAXB0sTeWgyT42HmBpPX2dKfQJBAMc
5Ml9nhAaFyM3dSMmDMbpGmEuBIoLzwXWYkvNB+EsChG6aW4Snsvnx6lCYY2rVR2eR1oLv+F8
UL3I2XEa5rmkCQBZXhnxnF9+Iei5y
/dKxpKYFFVVdCYOMFgtHMR42SHyD2Q8R6Dvpex2Ml4EYJULxr0TEqz6Z75M
/cMGSF9d9K2ECQQDEffSJoyjYwY2rGbPX8N5d9yrp3HLRbH4RjFGR0zCbSaA+PTQwxu2q1As
d8g7LN95UmyvliddJgayDIwnJSGse", "ctime": 1608044538 } ] }
```

The 2N devices allow up to four encryption keys to be used at one time. The first encryption key in the array is considered to be the primary encryption key and the other encryption keys are secondary. If a Bluetooth device authenticates itself with any secondary encryption key the 2N device will prompt the Bluetooth device to replace its encryption key with the primary encryption key. Because of this the newest encryption key should always be added to the beginning of the array.

If an array of a length shorter than 4 is submitted, the missing encryption keys are deleted (replaced with an empty object).

The key ***type*** is not mandatory. If the algorithm type is omitted, the 2N® device will automatically assume RSA (***rsa***).

The key *ctime* is not mandatory. If the creation time is omitted or invalid, the 2N device will display Jan 1st 1970 00:00:00 in the configuration web and will not return *ctime* for this encryption key.

Response

The response to a **GET** request is in the *application/json* format. The **result** object contains keys *location* and *keys*.

The response to a **PUT** request does not contain any details. E.g., if there is an invalid encryption key value, the key will not be written without any notification.

Table 2. Response to GET Request JSON Keys

Key	Typical Returned Values	Description
location	String	Location ID of a 2N device. The details are described in the Request section.
keys	Array of objects containing encryption keys	The array length is always 4 (empty objects are returned for the missing keys). The details and structure of objects in the array are described in the Request section.

Example of Response to GET Request


```
{ "success": true, "result": { "location": "54-1046-0745", "keys": [ { "
type": "rsa", "key": "MIICXAIBAAKBgQCXmIX1U7wGFW3FiDdhq7BIktIc4lg7X2IMxL
E83I75S3BRPL/7LCAefnMUJL0uyyFdeMpRoOVhVs
/iPfnYPNf4AiQO4lIQh8tSKDeat5IfXSMy9zXMyHeBOBg19R+
/uShyJsnLoJoB5MJDowwkOuSMIskK+dA17+3E
/Y+ujhhpCQIDAQABAoGAfzHOVAUp4cDhFbgxH5Y6lun5uZqAhXCGiEgQngxBOhJ97uuV+V0Q
pgVa8S/SPAzbtd2/g7YIQB
/i1OVDWJfUbEiuBhr6ZHwk5jfCfF0KkmTQtEBd4bbCz+FWyoy19DUXdsLNMf8GW4eWhooX+N
Cqc2sfloO4Nz+SpXmqpsMIc/ECQQDk63xVnRqPCgG3fqpLVGWkQl9wmYAIUP8MrdOAfRYfSC
/LrjX55lCRj4mAnSzRQfNclSEz2mITkoaCcjn1lTmXAkEAqYdjMEhIrg4LpYzqZDOF6v6w
/sUcLkepktTBYCFFV+YgOrlPr7akR8XtED8X/6QHwWciphp/5OBoJ
/KRAWWGXwJAZ0YNe5o6pxk+mQed0AotKKOA5w15A0d3KMMqzaag2k
/4sAzR8QGEi4aT4+AEngsAvV3R8tCsumO6JxNdLnu51QJAO6abzB1jFXtztajDwMYwVOORO9
P3eoFUtYmPEVgjoi4jIQabd2R4oZiPNaw9sYHyCKdVlcS1Q7+CZqv
/QdKLWQJBAKeoGxqcpDHvMtzgcSj3lZz0Z8dWmgtTF7QO5boHhxtZ1SEo3MvlicVue8U1tV2
XjUR6r7YueuusM9GxhBqr5YI=", "ctime": 1608047754 }, { "type": "rsa", "key
": "MIICXQIBAAKBgQCfyMHsTjPKf3Dv00gwMrQAR5UZrpt3tBy3kBvPv4R4o9H7Zzse7+yK
wfPTddKJQOL1IrCX06Zo8SZAMotjjpMy1M9K27ZB95YtAYiGLLRWeLAJUkL4gixgkHeS+T8u
QxLW7
/etqwU00uPmd94ZEZy226CHdKQW3zge2WEtuQ5oCwIDAQABAoGAZCp6RyUPGpahufZ9fpmKd
dJqCduH4paqmfhhNu8coHQyIqQoT9CgPKwxqhJmlVxz6rCAe+1WmNrz27LT5uluJKViU0XnL
V7FHG2smagjQ3rPepgOGcayphuiIlHikaBCafxnCRV
/E1Ifg08d1xK4cK858yMjpoEgDdEJi0R2qmECQQDXqtwGiXYSRnZzR90eCjrip6IIQqJuARE
91LOLyOhkPzCiPPf2IrT1JQsw6Tu0ZTm3NJzZ0VSEdZU6s2NcKHsnAkEAvap5GacBi9EZ9ls
iaQj
/dVA6LbUnBCo7qwRj7SUYw6ikCvmcLjdpjR80twj3FTAXB0sTeWgyT42HmBpPX2dKfQJBAMc
5Ml9nhAaFyM3dSMmDMbpGmEuBIoLzwXWYkvNB+EsChG6aW4Snsvnx6lCYY2rVR2eR1oLv+F8
UL3I2XEa5rmkCQBZXhnxnF9+Iei5y
/dKxpKYFFVVdCYOMFgtHMR42SHyD2Q8R6Dvpex2Ml4EYJULxr0TEqz6Z75M
/cMGSF9d9K2ECQQDEffSJoyjYwY2rGbPX8N5d9yrp3HLRbH4RjFGR0zCbSaA+PTQwxu2q1As
d8g7LN95UmyvliddJgayDIwnJSGse", "ctime": 1608046389 }, { "type": "rsa",
"key": "MIICXQIBAAKBgQCwWXVu2CNcUFgoqQBQ5NjaLJVEWuAFryK
/h9jfNe+qDufFS+itWsWfvyvMkUhhiidPCpgoOgqEipkYa0q3maPKPS4CJXZBFo++JSzsgw6
a/VxH0n8joHfJf6nIEcCGcuMAa
/HOEoOZq7uL7n2jTsyVnnDbYClXENh4Np9izSX23QIDAQABAoGBAI5iDFDMrfAw5p0dpqWpv
/SXnoUsIkg0mYeu9UlzUOgrVLKVkW22Jm3OelyWyKwIUaid0zBXfHp7NRTkO9V1dSnS5CnuO
73tye9MV5TeLqjMSBVCSPZWJK
//hu1VaRAL9UTZc+1e277lOB8c1Fup4uxR4b757brrclNKjT1U4Hh5AkEA4mFz+IrgTtdiLN
LQdww5B3ZELmaOl+lkYgC50hvqy2TDNGGiKGPqYmEd/4ySHBmaGnoh9ZFxnC
/ItrNEXBGdawJBAMdsOd2qDdbOSie2TpsGJs5eEUrLX6yW
/w+siO4SXczCTnJXckZyj79eEOcnrTRK+SuDsN+8+wmO3b9CZqxOxtcCQQCUkukOAfddRzaD
vIhc2YTERPZSjbSgNulO+LL8Fp5uht8mjb1jTNATaTHK+nMaRiNBpU6MYLxziVjtr5H56wWp
AkA0fXYVtEcEPTQjk8bI4yufsf7XMwSxxuTH2WAJWeg6lwJS8lVv2YOgmT
/VuAnM89b17ynFGlbxQxt21iF0RR/tAkA0nJMkbD3daWYAutdOQjzemae3O0O9r90NZ
/Khj5tQpv8gLe6EEpyFUNQNqodNoTmkIJJmPLlBjyx+zTspdE+C" }, { } ] } }
```

location is by default the serial number of a 2N device. Change it accordingly to add several devices to one location.

5.16 api lpr

The following subsections detail the HTTP functions available for the **api/lpr** service.

- 5.16.1 api lpr licenseplate
- 5.16.2 api lpr image

5.16.1 api lpr licenseplate

The **api/lpr/licenseplate** function is used for access control by license plate recognition.

Service and Privileges Groups

- Service group is Camera.
- Privileges group is License Plate Recognition.

Methods

- POST

Request

The request contains parameters in the **application/json** format.

Table 1. Request JSON Keys

Key Name	Mandatory	Expected Values	Default Value	Description
lprUuid	Yes	uuid	-	Uuid of the licence plate recognition event generated by the License Plate Recognition system.
accessPoint	Yes	0 or 1	-	Indicates whether a vehicle with the detected license plate is entering (0) or exiting (1) – this is important for Access Rules that are applied to the event in the 2N device.
plateText	Yes	license plate string	-	Text of a recognized license plate that is used to identify a user in the 2N device directory.

Key Name	Mandatory	Expected Values	Default Value	Description
plateImage	No	image encoded in base64	No image	The image in which the license plate was recognized. The size of the image data is limited to 256 kB.

Example of Request

Error rendering macro 'scroll-pdf-code' : null

Response

The response is in the **application/json** format.Table 2. Response JSON Keys

Key	Typical Returned Values	Description
success	`true`, `false`	The value is `true` when the request is processed successfully. When there is an error, the value is `false` and additional information is available in the `error` key.

Example of Response

```
{
    "success": true
}
```

There may occur various errors (e.g. missing mandatory parameter). When Error code 13 (parameter data are too big) is returned, the request was not processed and it is necessary to send the request again with a smaller image or without an image.

Subsequently received duplicate valid requests are ignored (the last ten successful requests are held in the memory). It is possible to attempt at resending a request when there is no reply from a 2N device without the risk of a duplicate barrier opening or duplicate event logging.

5.16.2 api lpr image

The **api/lpr/image** function is used for getting images received from the license plate recognition.

Service and Privileges Groups

- Service group is Camera.
- Privileges group is License Plate Recognition.

Methods

- GET

Request

The request contains parameters in the URL.

Table 1. Request Parameters

Parameter Name	Mandatory	Expected Values	Default Value	Description
plateText	Yes	String with license plate text	-	Text of a recognized license plate that is used to identify which image should be returned (up to five images are stored). If two or more images belong to the same license plate text, the newest one is returned.

Example of Request

```
URL: https://192.168.1.1/api/lpr/image?plateText=ABC123456
```

Response

The success response is in the **image/jpeg** format.

There may occur various errors (e.g. missing a mandatory parameter). Errors are returned in json with a response code 200. If there is no image associated to the submitted plate text, error 15 "no data available" is returned.

5.17 api accesspoint blocking

The following subsections detail the HTTP functions available for the **api/accesspoint/blocking** service.

- 5.17.1 api accesspoint blocking ctrl
- 5.17.2 api accesspoint blocking status

5.17.1 api accesspoint blocking ctrl

The **api/accesspoint/blocking/ctrl** function controls blocking of access on individual access points.

Service and Privileges Groups

- Service group is System.
- Privileges group is System Access Control.

Methods

- GET
- POST

Request

The request contains parameters in the URL.

Table 1. Request URL Parameters

Parameter Name	Mandatory	Expected Values	Default Value	Description
id	Yes	Integer (0, 1)	-	Specifies the identifier of the access point that is to be controlled (0 for Entry and 1 for Exit).
action	Yes	String (on, off)	-	Specifies whether the blocking for the corresponding access point should be switched on or switched off.

Example of Request

```
URL:
https://192.168.1.1/api/accesspoint/blocking/ctrl?id=0&action=on
```

Response

The response is in the **application/json** format.

Table 2. Response JSON Keys.

Key	Typical Returned Values	Description
success	`true`, `false`	The value is `true` when the request is processed successfully (i.e. the access blocking is in the desired state regardless of a change).

Example of a Response

```
{
    "success": true
}
```

There may occur various errors (e.g. missing mandatory parameter). When Error code 18 (access point disabled) is returned, the request was not processed because the specified access point was disabled at the time.

5.17.2 api accesspoint blocking status

The **api/accesspoint/blocking/status** function returns the status of access blocking for individual access points.

Service and Privileges Groups

- Service group is System.
- Privileges group is System Access Monitoring.

Methods

- GET
- POST

Request

The request contains parameters in the URL.

Table 1. Request URL Parameters

Parameter Name	Mandatory	Expected Values	Default Value	Description
id	No	Integer (0, 1)	All	Specifies for which access point the status should be returned. If this parameter is omitted, the access blocking status is returned for all the access points.

Example of Request

```
URL: https://192.168.1.1/api/accesspoint/blocking/status?id=0
```

Response

The response is in the **application/json** format. The value of the `result` key contains one key `accessPoints`, which contains an array with an object for each access point (the array has a length of 1 if an access point was specified in the request). The objects in the array contain the following keys. Table 2. Response JSON Keys

Key	Typical Returned Values	Description
id	Integer (0, 1)	Identifies the access point (0 for Entry, 1 for Exit).
blocked	Boolean (`true`, `false`)	Contains the current status of access point blocking (`true` when the access point is blocked, `false` when the access point is not blocked).

Example of Response


```
{ "success": true, "result": { "accessPoints": [ { "id": 0, "blocked": t
rue }, { "id": 1, "blocked": false } ] } }
```


There may occur various errors (e.g. insufficient privileges).

2N TELEKOMUNIKACE a.s.

Modřanská 621, 143 01 Prague 4, Czech Republic

Phone: +420 261 301 500, Fax: +420 261 301 599

E-mail: sales@2n.cz

Web: www.2n.cz

v2.33