2N® Helios IP HTTP API

Configuration Manual

Version 2.15

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® is a registered trademark of 2N TELEKOMUNIKACE a.s. Any product and/or other names mentioned herein are registered trademarks and/or trademarks or brands protected by law.

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...".

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 5

1.1 HTTP API Release Notes 6

2. HTTP API Description 7

2.1 HTTP Methods 9
2.2 Request Parameters 10
2.3 Replies to Requests 11

3. HTTP API Services Security 13

4. User Accounts 15

5. Overview of HTTP API Functions 18

5.1 api system info 22
5.2 api system status 23
5.3 api system restart 24
5.4 api firmware 25
5.5 api firmware apply 26
5.6 api config 27
5.7 api config factoryreset 28
5.8 api switch caps 29
5.9 api switch status 31
5.10 api switch ctrl 32
5.11 api io caps 33
5.12 api io status 34
5.13 api io ctrl 35
5.14 api phone status 36
5.15 api call status 37
5.16 api call dial 38
5.17 api call answer 39
5.18 api call hangup 40
5.19 api camera caps 41
5.20 api camera snapshot 43
5.21 api display caps 44
5.22 api display image 45
5.23 api log caps 46
5.24 api log subscribe 48
5.25 api log unsubscribe 50
5.26 api log pull 51
5.27 api audio test 65
5.28 api email send 66

5.29 api pcap 67
5.30 api pcap restart 68
5.31 api pcap stop 69

1. Introduction

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

2N® Helios IP 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 **2N® Helios IP 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 **2N® Helios IP HTTP API** demonstration and testing.

1.1 HTTP API Release Notes

Version	Changes
2.15	▪ Addition of new events: **TamperSwitchActivated**, **UnauthorizedDoorOpen**, **DoorOpenTooLong** and **LoginBlocked** ▪ Extension of events: **tzShift** 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** functi ons 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 a **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 ▪ Addition of 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** func tions are assined 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® Helios IP 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 **2N® Helios IP 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**.

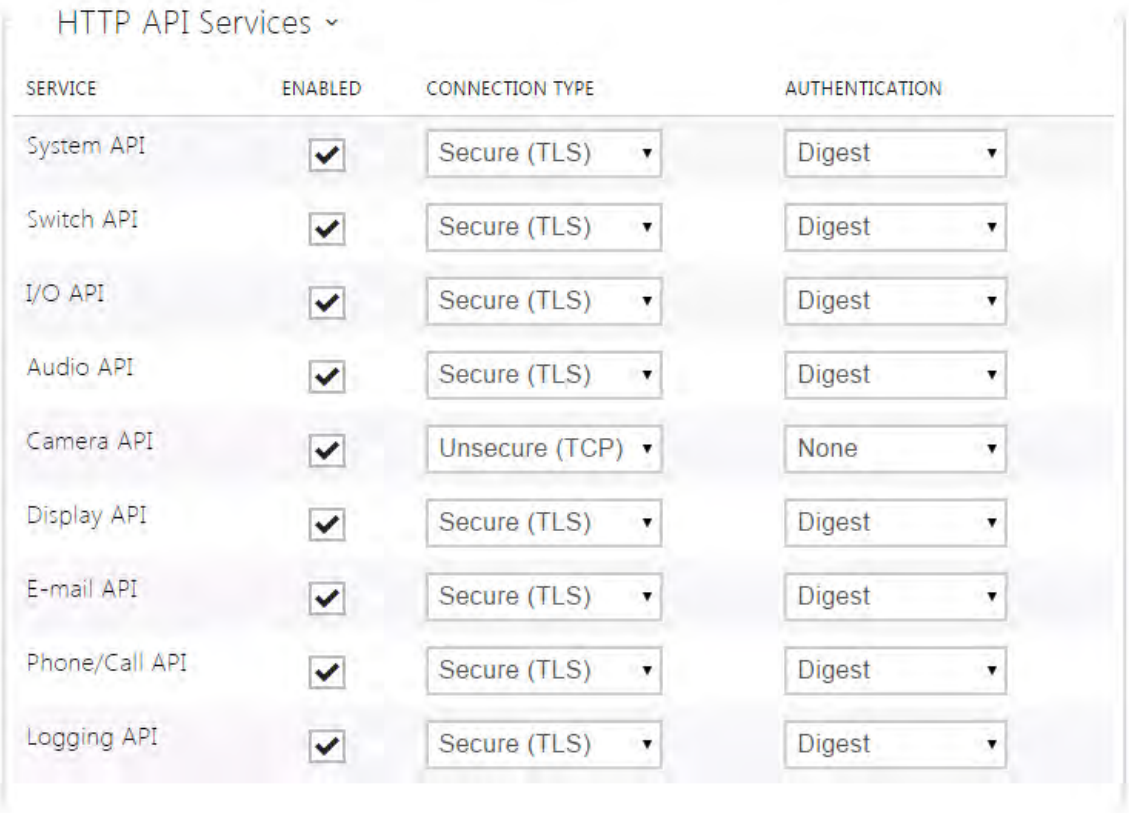
```
{
 "success" : false,
 "error" : {
   "code" : 12,
   "param" : "port",
   "description" : "invalid parameter value"
 }
}
```

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.
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.

3. HTTP API Services Security

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



HTTP API Services

SERVICE	ENABLED	CONNECTION TYPE	AUTHENTICATION
System API	✔	Secure (TLS)	Digest
Switch API	✔	Secure (TLS)	Digest
I/O API	✔	Secure (TLS)	Digest
Audio API	✔	Secure (TLS)	Digest
Camera API	✔	Unsecure (TCP)	None
Display API	✔	Secure (TLS)	Digest
E-mail API	✔	Secure (TLS)	Digest
Phone/Call API	✔	Secure (TLS)	Digest
Logging API	✔	Secure (TLS)	Digest

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.

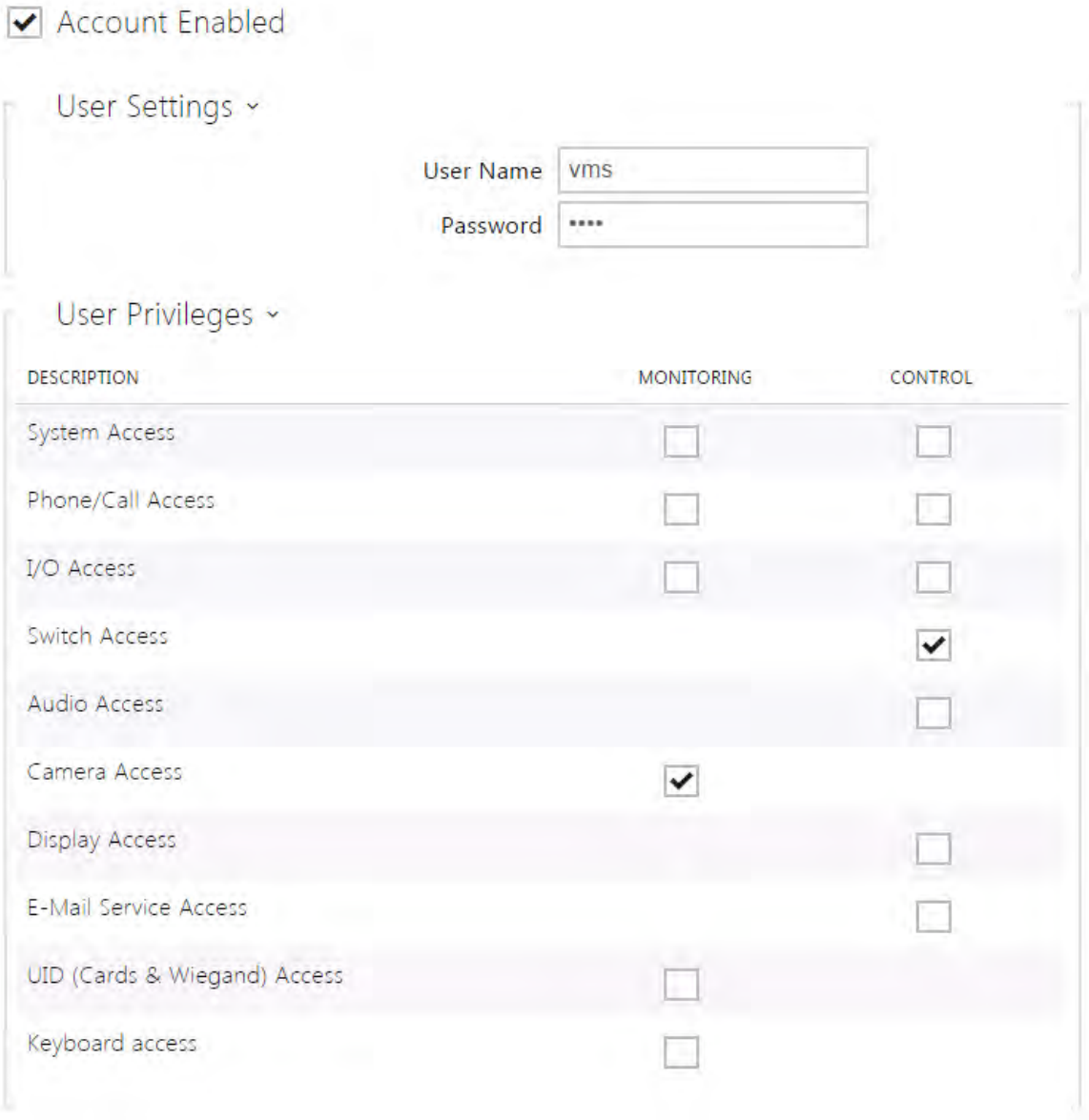
Set authentication methods for the requests to be sent to the intercom for each service. If the required authentication is not executed, the request will be rejected. Requests are authenticated via a standard authentication protocol described in **RFC-2617**. The following three authentication methods 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® Helios IP** 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/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
/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
/pcap	GET/POST	System	System Control	Yes
/pcap/restart	GET/POST	System	System Control	Yes
/pcap/stop	GET/POST	System	System Control	Yes

This chapter also includes:

- 5.1 api system info
- 5.2 api system status
- 5.3 api system restart
- 5.4 api firmware
- 5.5 api firmware apply
- 5.6 api config
- 5.7 api config factoryreset
- 5.8 api switch caps
- 5.9 api switch status
- 5.10 api switch ctrl
- 5.11 api io caps
- 5.12 api io status
- 5.13 api io ctrl
- 5.14 api phone status
- 5.15 api call status
- 5.16 api call dial
- 5.17 api call answer
- 5.18 api call hangup
- 5.19 api camera caps
- 5.20 api camera snapshot
- 5.21 api display caps
- 5.22 api display image
- 5.23 api log caps
- 5.24 api log subscribe
- 5.25 api log unsubscribe
- 5.26 api log pull
- 5.27 api audio test
- 5.28 api email send
- 5.29 api pcap
- 5.30 api pcap restart
- 5.31 api pcap stop

5.1 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"
 }
}
```

5.2 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
 }
}
```

5.3 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
}
```

5.4 api firmware

The **/api/firmware** function helps you upload a new firmware version to the device. When the upload is complete, use **/api/firmware/apply** to confirm restart and FW change.

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 **PUT** method can only be used for this function.

Request parameters:

Parameter	Description
blob-fw	Mandatory parameter including device FW

The reply is in the **application/json** format and includes information on the FW to be uploaded.

Parameter	Description
version	Firmware version to be uploaded
downgrade	Flag set if the FW to be uploaded is older than the current one

Example:

```
PUT /api/firmware
{
 "success" : true,
 "result" : {
  "version" : "2.10.0.19.2",
  "downgrade" : false
 }
}
```

If the FW file to be uploaded is corrupted or not intended for your device, the intercom returns error code 12 – invalid parameter value.

5.5 api firmware apply

The **/api/firmware/apply** function is used for earlier firmware upload (**PUT /api/firmware**) confirmation and subsequent device restart.

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/firmware/apply
{
  "success" : true
}
```

5.6 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 Helios 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
}
```

5.7 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
}
```

5.8 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 E nhanced 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
   }
  ]
 }
}
```

5.9 api switch status

The **/api/switch/status** function returns the current switch statuses. Define the switch in the optional **switch** parameter. If the **switch** parameter is not included, states 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). Use also **/api/switch/caps** to know the exact count of switches.

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

Example:

```
GET /api/switch/status
{
  "success" : true,
  "result" : {
    "switches" : [
      {
        "switch" : 1,
        "active" : false
      },
      {
        "switch" : 2,
        "active" : false
      },
      {
        "switch" : 3,
        "active" : false
      },
      {
        "switch" : 4,
        "active" : false
      }
    ]
  }
}
```

5.10 api switch ctrl

The **/api/switch/ctrl** function controls the switch statuses. The function has two mandatory parameters: **switch**, which determines the switch to be controlled, and **action**, defining the action to be executed over the switch (activation, deactivation, state change).

The function is part of the **Switch** service and the user must be assigned the **Switch 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
switch	Mandatory switch identifier (typically, 1 to 4). Use also **/api/switch/caps** to know the exact count of switches.
action	Mandatory action defining parameter (**on** – activate switch, **off** – deactivate switch, **trigger** – change switch state).
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/switch/ctrl?switch=1&action=trigger
{
 "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/switch/ctrl?switch=1&action=on&response=text
text
```

```
GET /api/switch/ctrl?switch=1&action=on&response=
```

5.11 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"
   }
  ]
 }
}
```

5.12 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
   }
  ]
 }
}
```

5.13 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=
```

5.14 api phone status

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
      }
    ]
  }
}
```

5.15 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"
   }
  ]
 }
```

5.16 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
 }
}
```

5.17 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
}
```

5.18 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
}
```

5.19 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"
   }
  ]
 }
}
```

5.20 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.

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
```

5.21 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
    }
   }
  ]
 }
}
```

5.22 api display image

The **/api/display/image** function helps you modify the content to be displayed: upload a GIF 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 a GIF image with display resolution (see **/api/display/caps**). The parameter is applied only if the **PUT** m ethod is used.

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

Example:

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

5.23 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 an increased noise level detection.	for microphone/microphone input equipped models only
KeyPressed	Signals pressing of a speed dial/numeric keypad button.	
KeyReleased	Signals releasing of a speed dial/numeric keypad button.	
CodeEntered	Signals entering of a user code via the numeric keypad.	for numeric keypad equipped models only
CardEntered	Signals tapping an RFID card on the card reader.	for RFID card reader equipped models only
InputChanged	Signals a change of the logic input state.	
OutputChanged	Signals a change of the logic output state.	
SwitchStateChanged	Signals a switch 1-4 state change.	
CallStateChanged	Signals a setup/end/change of the active call state.	
RegistrationStateChanged	Signals a change of the SIP server registration state.	
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.	

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"
  ]
 }
}
```

5.24 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 500 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

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** secon ds 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
 }
}
```

5.25 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,
}
```

5.26 api log pull

The **/api/log/pull** function 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.

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 (32-bit 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** e vent 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
code	string	User code, 1234, e.g.. The code includes 2 digits at least and **00** cannot be used.
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" : 23,
 "tzShift" : 0,
 "utcTime" : 1438072978,
 "upTime" : 9512,
 "event" : "CodeEntered",
 "params" : {
  "code" : "5864",
  "valid" : false
 }
}
```

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
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® Helios models) **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)
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" : 26,
 "tzShift" : 0,
 "utcTime" : 1438072979,
 "upTime" : 9513,
 "event" : "CardEntered",
 "params" : {
  "direction" : "any",
  "reader" : "ext7",
  "uid" : "01045E31BB",
  "valid" : false
 }
}
```


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

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
reason	string	Call end reason. The parameter is available only if the call end state is signalled. **normal** - normal call end **busy** - called station busy **rejected** - call rejected **noanswer** - no answer from called user **noresponse** - no response from called station (to SIP messages) **completed_elsewhere** - call answered by another station (group calls) **failure** - call setup failure
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 unauthorised 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	Unauthorised door opening state: **in** - signals the beginning of the unauthorised opening state. **out** - signals the end of the unauthorised 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"
 }
}
```

5.27 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
}
```

5.28 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 and 1.
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
}
```

5.29 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
```

5.30 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
}
```

5.31 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
}
```

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

HTTP API