An Axis company

Indoor Touch API

API guide

Version: 3.4.x

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.

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. Upgrade

Date	Version	Changes
01/04/2015	1.0	First version
04/05/2015	1.1	Hardware section added
08/03/2016	2.0.x	HTTP API section added

2. Purpose of Document

2N® IndoorTouch API (IA) is used for isolating the 2N® IndoorTouch specific functionality and helps share it with third party applications or can be used by a customer launcher.

2N® IndoorTouch HTTP API (HA) provides third party applications a possibility to configure and control selected system parts and the 2N® IP Mobile calling application.

Caution

- The HTTP API service is licensed under the 91378395 2N Indoor Touch HTTP API licence.

3. API

3.1 Architecture and Function

2N® IndoorTouch API is based on a daemon written in C, which forms a layer above the OS Linux core. It periodically executes such necessary actions as network configuration check, licence validity check, LED control, etc. and provides a function interface via the C library. It communicates with the library via the Unix socket. This library can be used by other C programs (for update and recovery modes, e.g.) too. The library is also included in Android API, where a Java interface android.hardware.IndoorTouch is created that makes it possible to call the library functions via the JNI. The interface in Java is the android.hardware.IndoorTouch class.

3.2 Hardware

3.2.1 Inputs and Outputs

3.2.1.1 HW v4-00-00

ID	Mark	Name	Restriction
AIN0/1	ANALOG IN 1/2	Analogue Input	0–5 V DC 1,5 M Ω
GP_OUT1/2	OUTPUT 1/2	Digital Output	Voh = 3,3 V DC 0,33 mA
GP_IN1/2	INPUT 1/2	Digital Input	Vihmax = 5 V DC 100 k Ω

GP_IO1/2	IN (OUT) 1/2	Digital Input/Output	Vihmax = 3,3 V DC
			(100 Ω serial resistor),
			Voh = 3,3 V DC
			4 mA (100 Ω serial resistor)

Relay0/1	REL1/2	Relay	contact data: max 30 V DC 2 A. DC 12 V +/- IN = power input: Vinmax = 12 V DC Imax = 1 A. DC 12 V + OUT = power output: Vout = 12 V DC Imax = 100 mA
	DC12V+ OUT	+ 12V output	Imax = 300 mA
	DC12V+ IN	+ 12V input	Imin = 1000 mA
	GND	Ground	

3.3 API Functions

The API functions are encased in the Java class android.hardware.IndoorTouch. All the methods are static and require no class instance. API is divided into:

- 3.3.1 LEDs
- 3.3.2 GPIO
- 3.3.3 System
- 3.3.4 Licence
- 3.3.5 Ethernet

3.3.1 LEDs

The LED notification interface has been designed both for GPIO and PWM LED control. Notifications are executed by so-called effects. The effect is a vector of brightness transitions for each defined LED colour with a defined transition time. With GPIO LEDs, intensity is thresholded for the on/off value. The library contains some pre-defined effects and the user can define additional effects of its own.

- 3.3.1.1 Effect Adding
- 3.3.1.2 Effect Removing
- 3.3.1.3 Effect Activation
- 3.3.1.4 Effect Deactivation
- 3.3.1.5 Effect Existence
- 3.3.1.6 Effect Enable
- 3.3.1.7 Display Backlight Notification

3.3.1.1 Effect Adding

```
int IndoorTouch.LedsAddEffect(String effectName, int continuity, String
transitions, int effectLedMask, int usedLedMask);
```

- 1. The argument specifies the effect name. If the effect exists, RC_ERR_EXIST is returned.
- 2. The argument can have the following values: LED_EFFECT_CONTINUITY_SINGLE, LED_EFFECT_CONTINUITY_REPEATING, LED_EFFECT_CONTINUITY_KEEP. These values define what happens when the transition vector ends. With SINGLE, the effect is terminated and removed from the active effect database. With REPEATING, the effect starts from the beginning again. With KEEP, the effect remains active displaying the last final transition value.

- 3. Transitions means a string of transitions in the following format: <initial brightness>,<target brightness>,<transition time in ms>, … . The count of numbers in the string must be divisible by 3 and contain one transition at least.
- 4. Mask of used LEDs for the effect itself. These LEDs are affected by the transition defined.
- 5. Mask of used LEDs. It contains all bits from the argument 4 mask plus others in which the brightness value is 0. The other LEDs are not affected by the effect and can use other effects that use a set of LEDs disjunctive with this effect. The following LEDs are available for effects at present:
  - LED_RED = 0x01;
  - LED_GREEN = 0x02
  - LED_BLUE = 0x04;
  - LED_NFC = 0x08;

Example of effect definition for red LED blinking:

```
IndoorTouch.LedsAddEffect("test-r", LED_EFFECT_CONTINUITY_REPEATING,"0,2
55,1000,255,0,1000", LED_RED, LED_RED | LED_GREEN | LED_BLUE));
```

The library includes a pre-defined effect "red-blink-missedcalls“ for red blinking.

The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_EXIST = -3; the effect name already exists
- IndoorTouch.RC_ERR_INVALID = -5; invalid function arguments
- IndoorTouch.RC_ERR_MAXREACHED = -10; maximum count of defined effects exceeded

3.3.1.2 Effect Removing

```
int IndoorTouch.LedsRemoveEffect(String effectName);
```

Removes an effect from the list of defined effects. If the so-defined effect is active, it is not affected by the call.
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_NOTEXIST = -4; the effect name does not exist

3.3.1.3 Effect Activation

```
int LedsActivateEffect(String effectName, int priority);
```

Activates an effect with a pre-defined priority: LED_PRIORITY_INFO=10, LED_PRIORITY_WARNING=20 or LED_PRIORITY_CRITICAL=30. A higher priority overrides a lower one. This means that if an effect with a higher priority is activated while an effect with a lower priority is active, the higher priority effect will be preferred. If the lower priority effect still remains after the higher priority effect ends, it will be activated again. Thus, the application does not have to be concerned with sharing LEDs with other applications - effect activation with an adequate priority is enough. By default, the blue LED backlight notification is activated automatically with priority 9.
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_NOTEXIST = -4; the effect name does not exist
- IndoorTouch.RC_ERR_DENIED = -7; notifications are disabled
- IndoorTouch.RC_ERR_MAXREACHED = -10; maximum count of defined effects exceeded
- IndoorTouch.RC_ERR_PRIORITY = -11; an effect with the same LED mask and identical priority has been started

3.3.1.4 Effect Deactivation

```
int IndoorTouch.LedsEffectDisable(String effectName);
```

Removes an active effect immediately. The LED brightness values remain on the level that was valid at the time of effect removal. If a lower priority effect is active controlling similar LEDs, the values will be reset immediately. As backlight notification is active in the RGB LEDs by default, the value is reset to black or blue after notification end.
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_NOTEXIST = -4; the effect name does not exist

3.3.1.5 Effect Existence

```
int IndoorTouch.LedsEffectExists(String effectName, int existence);
```

Returns the effect existence effectName in the selected existence database:

- LED_EFFECT_EXISTENCE_ACTIVE: active effects
- LED_EFFECT_EXISTENCE_DEFINED: defined effect vectors

The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_NOTEXIST = -4; the effect name does not exist in the selected database

3.3.1.6 Effect Enable

```
int IndoorTouch.LedsEffectsEnable(boolean enable);
```

Effects are enabled by default. If disabled, effect activation is ignored and all the effects are switched off except for display backlight notification.
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error

3.3.1.7 Display Backlight Notification

```
int IndoorTouch.LedsEnableBacklightNotification(boolean enable);
```

If enabled, the blue LED is displayed when the display is switched off (switched into the sleep mode).
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error

3.3.2 GPIO

3.3.2.1 GPIO Pin Reading

```
int IndoorTouch.GPIOGet(int pin);
```

Reads the current GPIO register value. Pin is one of the options:

- IndoorTouch.GPIO_PIN_IN1 = 0x01;
- IndoorTouch.GPIO_PIN_IN2 = 0x02;
- IndoorTouch.GPIO_PIN_OUT1 = 0x04;
- IndoorTouch.GPIO_PIN_OUT2 = 0x08;
- IndoorTouch.GPIO_PIN_IO1 = 0x10;
- IndoorTouch.GPIO_PIN_IO2 = 0x20;
- IndoorTouch.GPIO_PIN_RELAY1 = 0x40;
- IndoorTouch.GPIO_PIN_RELAY2 = 0x80;

The return value is {0,1} or

- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_SYS = -6; system error (the value cannot be read, etc.)
- IndoorTouch.RC_ERR_DENIED = -7; system error (GPIO does not exist)

3.3.2.2 GPIO Pin Setting

```
int IndoorTouch.GPIOSet(int pin, int value);
```

Pin is the PIN index, refer to **3.3.2.1 GPIO Pin Reading**, and value is {0,1}. The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_SYS = -6; system error (the value cannot be written, etc.)
- IndoorTouch.RC_ERR_DENIED = -7; GPIO does not exist or is unwriteable
- IndoorTouch.RC_ERR_LOCKED = -8; locked by another process via GPIOLock

3.3.2.3 GPIO Pin Locking

```
int IndoorTouch.GPIOLock(int pin);
```

Pin is the PIN index, refer to 3.3.2.1 GPIO Pin Reading. Locks the PIN against use with the active process. Another process can read the value but the setting ends with R C _ E R R _ L O C K E D .
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_LOCKED = -8; already locked by another process via GPIOLock

3.3.2.4 GPIO Pin Unlocking

```
int IndoorTouch.GPIOUnlock(int pin);
```

Pin is the PIN index, refer to 3.3.2.1 GPIO Pin Reading. Unlocks the selected PIN that was locked via GPIOLock.

The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_LOCKED = -8; already locked by another process via GPIOLock
- IndoorTouch.RC_ERR_NOTLOCKED = -9; PIN is not locked and need not be unlocked

3.3.2.5 Switchable GPIO Direction Getting

```
int IndoorTouch.GPIOGetDirection(int pin);
```

Pin is the PIN index, refer to **3.3.2.1 GPIO Pin Reading**. GPIO_PIN_IO1 and GPIO_PIN_IO2 can be I/O switched. This command helps you get the currently set direction. GPIO_DIRECTION_IN is set by default.

The return value is as follows:

- IndoorTouch.GPIO_DIRECTION_IN = 1;
- IndoorTouch.GPIO_DIRECTION_OUT = 2;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_SYS = -6; system error (the value cannot be read, etc.)
- IndoorTouch.RC_ERR_DENIED = -7; GPIO does not exist or is unswitchable

3.3.2.6 Switchable GPIO Direction Setting

```
int IndoorTouch.GPIOSetDirection(int pin, int direction);
```

Sets the direction of the switchable GPIO PIN. Pin is the PIN index, refer to **3.3.2.1 GPIO Pin Reading**. This switch is valid until the next ICU restart or reconfiguration. The state is not stored in the configuration.

Direction is one of the values:

- IndoorTouch.GPIO_DIRECTION_IN = 1;
- IndoorTouch.GPIO_DIRECTION_OUT = 2;

The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_SYS = -6; system error (the value cannot be set, etc.)
- IndoorTouch.RC_ERR_DENIED = -7; GPIO does not exist or is unswitchable

3.3.2.7 Active Low Pin Value Setting

```
int IndoorTouch.GPIOSetActiveLow(int pin, int value);
```

Pin is the PIN index, refer to **3.3.2.1 GPIO Pin Reading**, and value is {0,1}. Applicable for GPIO_PIN_IO1 and GPIO_PIN_IO2. The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error
- IndoorTouch.RC_ERR_SYS = -6; system error (the value cannot be set, etc.)
- IndoorTouch.RC_ERR_DENIED = -7; GPIO does not exist or is unswitchable

3.3.2.8 GPIO Input Value Change Detection

As the Java GUI interface is primarily controlled by messages and periodical reading of GPIO inputs is complicated for programmers and speed-limiting due to high JNI operational costs and delay caused by Java-daemon communication intervals, an auxiliary nested static class **IndoorTouch.GPIOWatcher** is available for detection of GPIO value changes. When its instance is created, **IndoorTouch.GPIOWatcher** starts a thread, which periodically checks the GPIO input states and, having detected a change, calls **public void OnGPIOChanged(int gpio, int value)** via Android Looper, i.e. the content of this method is executed in the UI thread. Hence, to monitor the GPIO state, the user creates a son of this class and overrides this method. Then the user creates an instance of this class. Example:

```
class MyWatcher extends IndoorTouch.GPIOWatcher {
    public MyWatcher(int mask) {
        super(mask);
    }
    @Override
    public void OnGPIOChanged(int gpio, int value) {
        if (gpio == IndoorTouch.GPIO_PIN_IN1) {
        // do the magic
        }
    }
}
protected IndoorTouch.GPIOWatcher mWatch = new MyWatcher( IndoorTouch.
GPIO_PIN_IN1 | IndoorTouch.GPIO_PIN_IN2);
```

GPIOWatcher has the following two constructors:

- public GPIOWatcher(int watchMask)

- public GPIOWatcher(int watchMask, int updateMs)

where watchMask is the GPIO pin bit array to be monitored and updateMs denotes the interval in miliseconds between the GPIO input checks. The default value is 200.

3.3.3 System

Upon the module start up, the serial console is set to the transmission rate of 115200 bps. Telnet is enabled on the Ethernet interface on port 2323 by default, which can be disabled by the launcher any time later.

- 3.3.3.1 CPU scaling
- 3.3.3.2 Display Uptime
- 3.3.3.3 Overheat detection
- 3.3.3.4 BootScreen Disable
- 3.3.3.5 Telnet

3.3.3.1 CPU scaling

The module periodically checks the display backlight. If the backlight is disabled and CPU scaling enabled, the module reduces the CPU rate from 1 Ghz to 300 Mhz. Enable /disable this functionality with the following command:

```
int IndoorTouch.SysEnableCPUDownscale(boolean enable);
```

Disable the function in the 'Display-Powersave CPU when display is off' menu in the launcher.
The return value is as follows:

- IndoorTouch.RC_OK = 0;
- IndoorTouch.RC_ERR_CONN = -1; API daemon connection error

3.3.3.2 Display Uptime

The module periodically checks the display backlight. If the backlight is enabled, the module counts the display backlight time in seconds, saving the value into the EEPROM every two minutes (saving the count of write cycles). The value is intended for technicians and can be obtained using the following Telnet command:

```
setpkey get hw_cfg.cfg.display_uptime (value in seconds).
```

To get the total IndoorTouch operation time enter the following command:

```
cat /proc/device_uptime (value in hours).
```

The function is always enabled and cannot be affected by the API.

3.3.3.3 Overheat detection

The overheat check is executed every 30 seconds. If the set temperature limit is exceeded, the display will be swiched off (and the processor underclocked if CPU scaling is enabled). There is a 5-seconds interval between overheat detection and display switch-off, which allows the GUI to display a device overheat warning. Set the display switch-off activation limit using the function below.

```
int IndoorTouch.SysSetMaxTemperature(float tempDeg);
```

where temperature in Celsius degrees is the argument. The GUI can get the current temperature value using the following function:

```
float IndoorTouch.SysGetDeviceTemperature();
```

The result is a temperature value in Celsius degrees. The GUI can detect device overheating by calling the following function periodically:

```
boolean IndoorTouch.SysIsOverheating();
```

3.3.3.4 BootScreen Disable

```
int IndoorTouch.SysDisableBootScreen();
```

The IndoorTouch launcher disables boot animation (a custom launcher must disable animation to display the system).

3.3.3.5 Telnet

Telnet is enabled by the library upon start up by default. The launcher can disable it as configured. Enable/disable Telnet as follows:

```
boolean IndoorTouch.SysEnableTelnet(boolean enable);
```

3.3.4 Licence

3.3.4.1 Licence Key Decoding

```
LicInfo IndoorTouch.LicUnpackProductKey(String productCode, String
serial);
```

ProductCode is a key with alphanumeric characters in the following format: XXXXX-XXXXX-XXXXX-XXXXX-XXXXX.
Serial provides the serial number assigned to the product key in the following format: ##-####-####.

The function unpacks information into the IndoorTouch.LicInfo static class:

```
static public class LicInfo {
    public boolean valid;
    public int hours;
    public int flags;
}
```

Valid defines that the licence key is valid for the selected serial number. Hours defines the count of licensed hours: 0 means no hour limit. Flags denotes licence attributes: 1 means a possibility to install customer applications.
The function allows 10 calls in 5 seconds at most. Further 10 calls are allowed in the next 5 seconds. When all the attempts are exhausted or a system error occurs, the method returns null.

3.3.4.2 Product Key Getting

```
String IndoorTouch.LicGetProductKey();
```

3.3.4.3 Serial Number Getting

```
String IndoorTouch.LicGetSerialNumber();
```

3.3.4.4 Product Key Validity Detection

```
boolean IndoorTouch.LicIsCurrentLicenceValid();
```

3.3.4.5 Product Key Setting

```
boolean IndoorTouch.LicSetProductKey(String productKey);
```

The method sets the product key into the 2N® IndoorTouch EEPROM. If the key is not valid for the 2N® IndoorTouch serial number or a system error occurs, the method returns false. Otherwise, true is returned.

3.3.5 Ethernet

As Android JB basic version supports only DHCP Ethernet setting and there are no such support and UI settings options in Android API, Ethernet will be configured via a special interface until an official support is available or an own system support is implemented.

- 3.3.5.1 Network Info Getting
- 3.3.5.2 DHCP Active Network Setting
- 3.3.5.3 Network Static Address Setting
- 3.3.5.4 Ethernet Interface Deactivation

3.3.5.1 Network Info Getting

```
NetInfo IndoorTouch.NetEthGetInfo();
```

The function returns the static class IndoorTouch.NetInfo:

```
static public class NetInfo {
    public boolean active;
    public boolean dhcpUsed;
    public String ip;
    public String mask;
    public String dns1, dns2;
    public String gateway;
}
```

- Active – network is active (cable connected)
- DhcpUsed – address is static/dynamic via DHCP
- ip
- mask
- dns1
- dns2 – may/may not be included. gateway

If a system error occurs, the method will return **null**.

3.3.5.2 DHCP Active Network Setting

The call activates the Ethernet interface with a dynamically retrieved address.

```
int IndoorTouch.NetEthSetDHCP()
```

3.3.5.3 Network Static Address Setting

The call activates the Ethernet interface with statically defined network parameters.

```
int IndoorTouch.NetEthSetStatic(String ip, String mask, String dns1,
String gateway);
```

3.3.5.4 Ethernet Interface Deactivation

```
int IndoorTouch.NetEthSetOff();
```

3.4 Installation

Simply import a Java library into applications that are part of the AOSP (import android.hardware.IndoorTouch) compiling it as part of the Android system.

With externally developed applications in Eclipse or AndroidStudio, include the IndoorTouch.java source code file to your project, using path src/java/android/hardware/IndoorTouch.java. After that import this file to your source code using import android.hardware.IndoorTouch. You will be able to use all functions after this. Source code file IndoorTouch.java can be downloaded here.

Debug directly on the 2N® IndoorTouch or emulate the library functions.

4. HTTP API

4.1 Interface

Communication is made via the HTTPS protocol, which runs on standardised port 443. Authentication requires login and password data to be transmitted in the first connection. Subsequently, the server returns the client a cookie with a session key for further requests. Having failed to send more requests within a timeout (20 minutes at present), the client will be logged out automatically. The next requests will thus return the HTTP return code 401 (Not authorised). Then the client has to log in again.

4.1.1 Requests

The client-server communication is used. The client sends a data/command execution request and the server sends a response. The request consists of URL and optional data in the following format:

```
https://<IP>/api/v1/<subsystem>?<parameters>
```

- IP is the IDT IP address
- <subsystem> designates the system part to be controlled
- <parameters> includes request details

For example:

```
https://192.168.254.228/api/v1/gpio?
action=set&name=io2&type=direction&value=out
```

means that the client wants to switch the GPIO pin to output in the GPIO subsystem.

If the data to be transferred are rather extensive, store the data in the JSON format in the HTTP request body (Content-Type: application/json; charset=UTF-8). Refer to the list of command parameters for these exceptions.

The HTTP method GET is used for the requests that do not transmit data. The PUT method is used for the requests that include data.

4.1.2 Responses to Requests

A response to a request always includes JSON data in the following format:

```
{
    "data": {
        "success": true,
        "array": [
            {
                "serial_number": "54-0562-0293"
            }
        ]
    }
}
```

Make sure that the cover-all **data** container and the nested boolean **success** are always included. Success is always true if the command has been (or is expected to have been) executed successfully. Nevertheless, it cannot be guaranteed for the time being that the success=true value really means success for all commands. For example, there may be lack of success information for the commands that **2N® Helios IP Mobile** resends to HIP and the value cannot be reported. Such commands include the PUT method and the commands that do not return values: calls, opening doors, etc. Thus, the user has to use another request (call/status, e.g.) to know the command's success.

The false value is returned for many reasons. At present, the HTTP API cannot distinguish errors and specific error messages will be implemented in the next versions if necessary. The short list of potential errors is as follows:

- Request entering error – invalid argument values, missing arguments, non-existent subsystem, wrong JSON data format, incorrect HTTP method, etc.
- A **2N® Helios IP Mobile** command is entered but the application is not running.

  Therefore, make sure that the application has been started before sending a **2N® Helios IP Mobile** command.

The **data** container can include more items whose names and contents depend on the request to be sent. Our example includes an array of serial numbers.

4.1.3 Cautions

Currently, API is used for both providing data to the web interface (REST) and user HTTP API requirements. Thus, the data that can be viewed and configured via the web can be available in API despite not being documented.
API now supports just one web interface login at a time. Hence, if a web interface is

open and the user logs in to another one, API will log the user out of the first one. This will become obvious when the user enters another request via the web and is forwarded to the login dialogue. This means that the web is used by another user and data might be overwritten.

HTTP API has no such limitation. This means that more users may be authenticated, but mutual data overwriting is not checked and has to be solved by the developer.

4.2 Commands

- 4.2.1 Session Control
- 4.2.2 GPIO
- 4.2.3 2N® IP Mobile Control
- 4.2.4 System

4.2.1 Session Control

- 4.2.1.1 Login
- 4.2.1.2 Logout

4.2.1.1 Login

The login helps you get access to the selected HTTPS session using a username and password. The login is Admin by default and the password is identical with the 2N launcher configuration access password.

URL:	https://<IP>/api/v1/login
HTTP method:	PUT
Request body:	{"login":"<username>","password":"<password>"}
Response code:	200 if everything is OK, otherwise 401,405,...
Response data:	N/A

In case the solution requires the **2N® Indoor Touch** input/output control from **2N Helios IP**, you can use an unsecured login to **2N® Indoor Touch**. Set the parameters in the Launcher settings: in the **Launcher Configuration,** Software section or the web **2N® Indoor Touch**, Device / Maintenance section. Having enabled this option, you can use the following URL for login:

```
https://<IP>/api/v1/login?login=Admin&password=<set_password>
```

Example of use:

```
Event.KeyPressed Key=1
Action.SendHttpRequest uri=https://10.27.1.32/api/v1/login?
login=Admin&password=2n; event=1
```

As a result, **2N Helios IP** logs in to **2N® Indoor Touch** HTTP API when you press button 1.

4.2.1.2 Logout

The logout helps you log out the currently logged-in session.

URL:	http://<IP>/api/v1/logout
HTTP method:	GET
Request body:	N/A
Response code:	200 if everything is OK, otherwise 401,405,...
Response data:	N/A

4.2.2 GPIO

- 4.2.2.1 GPIO List Getting
- 4.2.2.2 Input Value Getting
- 4.2.2.3 Output Value Setting

4.2.2.1 GPIO List Getting

The list of GPIOs (eight at present) stored in the array container is returned. Name {in1, in2, out1, out2, io1, io2, relay1, relay2} is the input identifier that is subsequently used for GPIO identification in the HTTP API communication too. Direction takes the values {in, out, io}.

URL:	https://<IP>/api/v1/gpio/caps
HTTP method:	GET
Request body:	N/A

Response code:	Standard HTTP response code, treating syntactic errors

Response data:	
	```
	{
	    "data": {
	        "success": <boolean>,
	        "array": [
	            {
	                "direction": "<direction>
	",
	                "index": <i/o index>,
	                "name": "<i/o
	identifier>"
	            },
	            ...
	        ]
	    }
	}
	```

4.2.2.2 Input Value Getting

Get the input value and/or switchable IO direction. Name – GPIO name, refer to **4.2.2.1 Získání seznamu GPIO vstupů a výstupů**. Type can have the values {value, direction}.

GPIO name {in1, in2}, type=value:

- Value can be {0, 1}

GPIO name {io1, io2}, type = direction:

- Value can be {in, out}

URL:	`https://<IP>/api/v1/gpio?action=get&name=<string>&type=<string>`
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors

Response data:	`{"data":{"value":<bit>,"success":<boolean>}}`

4.2.2.3 Output Value Setting

Set the output value or switchable IO direction. Name – GPIO name, refer to **4.2.2.1 GPIO List Getting**. Type can have the values {value, direction}.

GPIO name {out1, out2, relay1, relay2}, type=value:

- Value can be {0, 1}

GPIO name {io1, io2}, type = direction:

- Value can be {in, out}

Success can be false if GPIO is locked for setting via another application.

URL:	https://<IP>/api/v1/gpio?action=set&name=<string>&type=<string>&value=<bit>
HTTP method:	GET
Request body:	N/A
Response code:	
Response data:	{"data":{"success":<boolean>}}

4.2.3 2N® IP Mobile Control

- 4.2.3.1 Call Dialling
- 4.2.3.2 Call Accept
- 4.2.3.3 Call Termination
- 4.2.3.4 Call State
- 4.2.3.5 2N IP Intercom List Getting
- 4.2.3.6 Getting 2N IP intercoms Added to Devices
- 4.2.3.7 Change of Items in Added Device List
- 4.2.3.8 Door Lock Activation
- 4.2.3.9 2N® IP Mobile Minimisation
- 4.2.3.10 2N® IP Mobile State
- 4.2.3.11 2N® IP Mobile Restart

4.2.3.1 Call Dialling

URL:	https://<IP>/api/v1/call/dial?number=<sip_address>
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true}}

4.2.3.2 Call Accept

Accepts currently ringing incoming call.

URL:	https://<IP>/api/v1/call/accept
HTTP Metoda:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true}}

4.2.3.3 Call Termination

Terminates the active call.

URL:	https://<IP>/api/v1/call/hangup
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true}}

4.2.3.4 Call State

Get the current state of the call processing layer as one of the strings below:

```
{Idle, IncomingReceived, OutgoingInit, OutgoingProgress,
OutgoingRinging, OutgoingEarlyMedia, Connected, StreamsRunning,
Pausing, Paused, Resuming, Referred, Error, CallEnd, PausedByRemote,
UpdatedByRemote, IncomingEarlyMedia, Updating, Released,
EarlyUpdatedByRemote, EarlyUpdating}
```

Refer to http://www.linphone.org for more information.

URL:	https://<IP>/api/v1/call/status
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":<boolean>, "call_state":"<string>"}}

4.2.3.5 2N IP Intercom List Getting

Get the list of the **2N IP intercoms** that are registered with **2N® IP Mobile** and can be displayed and edited in the device list.

NNURL:	https://<IP>/api/v1/devices?action=get&type=all
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors

Response data:	
	```
	{
	    "data": {
	        "success": <boolean>,
	        "array": [
	            {
	                "ip": "192.168.254.222",
	                "sip_uri": "sip:
	192.168.254.222",
	                "serial_number": "54-
	0562-0293",
	                "name": "2N IP Vario"
	            }
	        ]
	    }
	}
	```

4.2.3.6 Getting 2N IP intercoms Added to Devices

Get the list of the **2N IP intercoms** added to the list of devices.

URL:	https://<IP>/api/v1/devices?action=get&type=add
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors

Response data:	
	```
	{
	    "data": {
	        "success": true,
	        "array": [
	            {
	                "serial_number": "54-
	0562-0293"
	            }
	        ]
	    }
	}
	```

4.2.3.7 Change of Items in Added Device List

Modify the list of added devices. The value to_add includes the serial numbers separated with commas (,) to be added to the list. Analogously, to_remove defines the serial numbers to be removed.

URL:	https://<IP>/api/v1/devices?action=put
HTTP method:	PUT
Request body:	{"to_add":"","to_remove":"54-0562-0293"}
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	N/A

4.2.3.8 Door Lock Activation

Send a request to activate the door lock for a selected 2N IP intercom. Make sure that the 2N IP intercom serial number is in the list of added devices.

URL:	https://<IP>/api/v1/door?action=open&serial=<string>&switchid=<integer>
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true}}

4.2.3.9 2N® IP Mobile Minimisation

Send the 2N® IP Mobile application to the background (displaying the 2N® Indoor Touch launcher).

URL:	https://<IP>/api/v1/hipm?action=minimize
HTTP method:	GET

Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true}}

4.2.3.10 2N® IP Mobile State

Make sure that the application is running and capable of processing data.

URL:	https://<IP>/api/v1/hipm?action=running
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true, "running":<boolean>}}

4.2.3.11 2N® IP Mobile Restart

Restart **2N® IP Mobile** if running and start **2N® IP Mobile** if not running. Once the application is started, the system checks it automatically and restarts it when necessary.

URL:	https://<IP>/api/v1/hipm?action=restart
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	{"data":{"success":true}}

4.2.4 System

- 4.2.4.1 System Info and Status
- 4.2.4.2 2N® Indoor Touch Restart

4.2.4.1 System Info and Status

As this is a WEB API function, the success value is not included in the response. Information on the system and its state is returned.

URL:	https://<IP>/api/v1/hipm/status
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treating syntactic errors
Response data:	"{"data":{"uptimeD":0,"totalStorage":"1112 MB", "hw":"3", "fw":"2.0.0.4.0", "installed":" 512 MB", "state":{"id":4}, "date":"4\/14\/2016", "remainingHours":"-2", "nfccapable": true, "temperature":"0.0 °C", "uptimeHours":"4012 hours", "time":"10:51AM", "totalCache":"92 MB", ":key":"J52RF-NVDD4-YW3P4-WCM9K-2TAIM", "os":"Android OS 4.2.2", "sn":"99-9999-9999", "lanmac":"D0-39-72-1F-56-C8", "freeStorage":"989 MB", "totalMemory":"496 MB", "keys":"APP LAUNCHER ROOT", "freeMemory":"363 MB", "freeCache":"88 MB", "system":"2N® Indoor Touch", "wifimac":"D0-39-72-1F-56-CA", "uptimeM":9, "wificapable":true, "uptimeH":0}}".

Meaning of items:

- uptimeD – count of days for which IDT has been working and available, see uptimeH, uptimeM
- totalStorage – total data storage size
- hw – IDT hardware version
- fw – firmware version
- installed – available IDT installed internal memory size
- state/id – licence state/ID, namely:
  - invalid key product
  - licence has limited operation hours and the limit has been exceeded
  - licence has limited operation hours and/or the limit has not been exceeded (limit ok)
  - licence licence has not limited operation hours
- date – current IDT date

- remainingHours – count of remaining hours for a licence with limited operation time or:
    - licence invalid
    - operation time limit exceeded
    - limit not set
    - other error
- nfccapable – true if NFC support is available
- temperature – IDT temperature
- uptimeHours – count of hours since the IDT restart
- time – current IDT time
- totalCache – cache storage total size
- key – IDT product key
- os – operating system version
- sn – IDT serial number
- lanmac – Ethernet interface MAC address
- freestorage – free data storage size
- totalMemory – total available internal memory size
- keys – licence keys for additional IDT functions, namely:
    - APP – possibility to install additional applications
    - LAUNCHER – possibility to switch to classic Android launcher
    - ROOT – possibility to set root rights to selected applications
    - HTTPAPI – possibility to control IDT via API
- freeMemory – free internal memory size
- freeCache – free cache storage size
- system – device name
- wifimac – WiFi interface MAC address
- uptimeM – count of minutes for which IDT has been working and available , see uptimeD, uptimeH
- wificapable – true if IDT is equipped with a WiFi interface
- uptimeH – count of hours for which IDT has been working and available , see uptimeD, uptimeM

4.2.4.2 2N® Indoor Touch Restart

Restart the **2N® Indoor Touch** unit.

URL:	https://<IP>/api/v1/maintenance/resetdevice
HTTP method:	GET
Request body:	N/A
Response code:	Standard HTTP response code, treatting syntactic errors
Response data:	N/A

5. Supplementary Information

5.1 Directives, Laws and Regulation - General Instructions and Cautions

Please read this User Manual carefully before using the product. Follow all instructions and recommendations included herein.

Any use of the product that is in contradiction with the instructions provided herein may result in malfunction, damage or destruction of the product.

The manufacturer shall not be liable and responsible for any damage incurred as a result of a use of the product other than that included herein, namely undue application and disobedience of the recommendations and warnings in contradiction herewith.

Any use or connection of the product other than those included herein shall be considered undue and the manufacturer shall not be liable for any consequences arisen as a result of such misconduct.

Moreover, the manufacturer shall not be liable for any damage or destruction of the product incurred as a result of misplacement, incompetent installation and/or undue operation and use of the product in contradiction herewith.

The manufacturer assumes no responsibility for any malfunction, damage or destruction of the product caused by incompetent replacement of parts or due to the use of reproduction parts or components.

The manufacturer shall not be liable and responsible for any loss or damage incurred as a result of a natural disaster or any other unfavourable natural condition.

The manufacturer shall not be held liable for any damage of the product arising during the shipping thereof.

The manufacturer shall not make any warrant with regard to data loss or damage.

The manufacturer shall not be liable and responsible for any direct or indirect damage incurred as a result of a use of the product in contradiction herewith or a failure of the product due to a use in contradiction herewith.

All applicable legal regulations concerning the product installation and use as well as provisions of technical standards on electric installations have to be obeyed. The manufacturer shall not be liable and responsible for damage or destruction of the product or damage incurred by the consumer in case the product is used and handled contrary to the said regulations and provisions.

The consumer shall, at its own expense, obtain software protection of the product. The manufacturer shall not be held liable and responsible for any damage incurred as a result of the use of deficient or substandard security software.

The consumer shall, without delay, change the access password for the product after installation. The manufacturer shall not be held liable or responsible for any damage incurred by the consumer in connection with the use of the original password.

The manufacturer also assumes no responsibility for additional costs incurred by the consumer as a result of making calls using a line with an increased tariff.

Electric Waste and Used Battery Pack Handling

Do not place used electric devices and battery packs into municipal waste containers. An undue disposal thereof might impair the environment!

Deliver your expired electric appliances and battery packs removed from them to dedicated dumpsites or containers or give them back to the dealer or manufacturer for environmental-friendly disposal. The dealer or manufacturer shall take the product back free of charge and without requiring another purchase. Make sure that the devices to be disposed of are complete.

Do not throw battery packs into fire. Battery packs may not be taken into parts or short-circuited either.

An Axis company

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

v3.4.x