

NEXO NLX₁ SYSTEM



INSTRUCTIONS MANUAL

VERSION 2.0



ojmar

INTELLIGENT LOCKING SYSTEMS

W S D



NEXO NLX₁ SYSTEM



INSTRUCTIONS MANUAL

VERSION 1.0



INTELLIGENT LOCKING SYSTEMS

This document is for information only and is not contractually binding.

The information may have undergone changes that have not yet been incorporated in the present document and we therefore suggest that if in doubt you please contact Ojmar to obtain updated information.

© Ojmar, S.A.

CONTENTS

1. INTRODUCTION	8
1.1 SYSTEM ELEMENTS.....	8
1.2 OPERATION	9
1.3 GENERAL WARNINGS.....	10
1.3.1 Regulatory information USA	10
1.3.2 Class B device notice	11
1.3.3 RF exposure safety.....	11
1.4 TECHNICAL SUPPORT	11
2. INSTALLATION MANUAL	12
2.1 SDK INSTALLATION	12
2.1.1 Installation	12
2.2 PHYSICAL INSTALLATION OF THE ROUTER.....	28
2.3 ASSEMBLY OF THE NEXO NLX ₁ LOCK	28
2.3.1 Lock Direction.....	29
2.3.2 Assembly Considerations	29
2.3.3 Steps to Follow for Assembly.....	34
3. START-UP	40
3.1 CENTRAL PC AND PCS (IPS, NETWORK, ETC.).....	40
3.2 READER/PORTABLE PROGRAMMER.....	43
3.2.1 Ojmar NFC Model Programmer	44
3.2.2 Desktop Reader	50
3.3 LOADING INITIALISATION DATA TO THE NFC PROGRAMMER	51
3.4 INITIALISE LOCKS WITH NFC PROGRAMMER	51
3.5 DESCRIPTION OF LED FLASHES	55
3.5.1 Lists	56
4. CARD RECORDING.....	58
4.1 FREE	58
4.2 FIXED.....	58
4.3 MASTER.....	58
4.4 SERVICE.....	59
4.5 DELETED.....	59
5. MAINTENANCE MANUAL	60
5.1 OPENING OF THE NEXO NLX ₁ LOCK USING EMERGENCY POWER SUPPLY	60
5.2 EMERGENCY MECHANICAL OPENING OF THE NEXO NLX ₁ LOCK.....	63
5.3 DISASSEMBLY OF THE NEXO NLX ₁ LOCK	65
5.4 REQUEST FOR DATA: EVENTS, CYCLES, SETUP.....	67

5.5	BATTERY REPLACEMENT	68
5.6	CLEANING.....	69
5.7	FAQ	71
5.8	DECLARATION OF CONFORMITY	72
6.	API INTRODUCTION	75
6.1	PURPOSE OF THE DOCUMENT	75
6.2	SYSTEM INTRODUCTION.....	75
6.2.1	NEXO NLX1 Particularities	76
6.3	API STRUCTURE OVERVIEW.....	78
6.3.1	Online Communications.....	79
6.4	DOCUMENTATION PROVIDED	80
6.5	SYSTEMS WORKFLOW	80
6.5.1	Device Configuration	80
6.5.2	Customers' usage	81
6.5.3	Devices' maintenance	81
6.6	LOCK CONFIGURATION EXAMPLE	83
6.6.1	Configured by RFID cards for OTS family	83
6.6.2	Configured by NFC programmer for OTS family.....	84
6.6.3	Set up via NFC for Nexo NLX1 family.....	85
6.6.4	Configured via TCP/IP for Nexo NLX1 family	86
7.	API REFERENCE MANUAL	87
7.1	INTRODUCTION	87
7.2	API REFERENCES INDEX	88
7.3	API REFERENCES DEFINITIONS.....	89
7.3.1	Datamodel's API.....	89
7.3.2	Recorder's API	121
7.3.3	Socket API.....	131
7.4	EXAMPLE OF HOW TO CONFIGURE LOCK FROM FACTORY MODE	169
7.4.1	Lock Complete Configuration.....	169
7.4.2	User card request	172
7.5	CODE EXAMPLES.....	174
7.5.1	PHP	174
7.5.2	Java	175
7.5.3	C.....	177
7.6	ANEXO.....	179
7.6.1	Generate and read JSON files.....	179
7.6.2	How to make a REST call.....	180
8.	ANNEXES	185
8.1	PHYSICAL INSTALLATION OF THE ROUTER.....	185

PAGE LEFT BLANK INTENTIONALLY

1. INTRODUCTION

Nexo NLX1 is an electronic locking system mainly designed for sports and leisure centres and schools. The system operates with RFID proximity media (Wellness® wristbands, cards and key) with technologies based on international standards.

The Nexo NLX1 system connects wirelessly to the central PC of the system and that way can:

- Configure the locks quickly and conveniently.
- Manage the status of the lockers in real time.

Retrofit coin locks are very easy as they have the same anchor holes.

The locks comply with all standards anchorage and dimensions and therefore the old locking systems can be replaced without having to change the cupboards or lockers.

- NB.: The locks must be initialized beforehand.

1.1 SYSTEM ELEMENTS

The complete system consists of the following components:

- Nexus NLX1 Locks (Assembly in Section 2.3 and Disassembly in Section 5.3) including the strike plate (depends on the design of the locker)
- Wi-Fi Router (see Section 2.2).
- PC (see Section 3.1).
- Desktop reader for media (Wellness® Key, cards, wristbands) and/or Ojmar NFC portable programmer (see Section 3.2).
- Maintenance kit (see Section 4).
- Emergency power supply tool.
- Emergency mechanical opening tool.API for Management software

1.2 OPERATION

To start up the Nexo NLX1 system, the steps detailed below must be followed:

1. Configure the PC (see Section 3.1).
2. Configure the router (see Section 2.2).
3. Configure the parameters of the NFC programmer (IP, network, locks...) connecting it to the PC and using the software (see Section 3.3).
4. Initialize each lock with the NFC programmer. This is done only the first time, for the start-up of the system (see Section 3.4).

• NB.: Once initialized, the lock connects to the PC wirelessly, sending back signal confirming connection to the system

5. Write the keys with the NFC programmer or with the CNReader card reader (see Section 4).

There are different types of keys; free user, fixed user, master, service and reset (see Section 4).

The system has two types of user:

- User: The user opens or closes the locks with one or several keys according to the permissions assigned to the user.
- Manager: Gives permissions to the cards, administers the keys that open the locks and asks the locks for information (see Section 5.4).

The system has two operating modes:

- On-line (normal operation of the system):
 - User's key makes contact with front part of the lock.
 - The lock informs the PC that a card is trying to open/close it.
 - The PC decides whether to open/close the lock according to the permissions that have been given to the user/key.
 - After performing the operation (allow opening/closing the lock or not), the event is sent to the PC with the battery level of the lock (see Section 5.4).
- NB.: The communication starts with the card or the programmer; the PC cannot initiate communication.
- Off-line (when occasionally wireless connection is not available):
 - The user puts the key in front of the front part of the lock.
 - The lock attempts to connect to the PC and, when it cannot, it acts according to the last configuration it has received from the latter when the system was on-line.
 - When the connection is reset and it connects, the PC requests information to the lock of what has happened during the time that it was off-line.

1.3 GENERAL WARNINGS

To ensure the correct operation of our product in your facilities you must follow the following rules:

- Product installation and use must be carried out in accordance with the technical operating conditions described in the corresponding manual.
- When not specifically indicated, the proper installation and use of the application is the responsibility of the customer.
- Inspect the packaging and material for damage immediately after reception of the material. Also check that the delivery is complete (accessories, documentation, etc.).
- If the packaging has been damaged during transport or you suspect that it could have been damaged or may be faulty, the material must not be started up. In this case, please contact us.
- Our products' installation and handling must be carried out by authorised staff. The electrical connections in particular must be carried out only by qualified specialists.
- Any replacement or removal of the protection covers is strictly forbidden.
- Do not attempt to repair materials after a fault or damage and try to operate it again. In such an event, it is essential you contact us.
- We take no responsibility for damage caused as the result of misuse.
- Before leaving the devices in their pick-up installations, the batteries must be removed and left separately for the proper management.



1.3.1 Regulatory information USA

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

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

1.3.2 Class B device notice

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

1.3.3 RF exposure safety

This product is a radio transmitter and receiver. The antenna must be installed and operated with minimum distance of 20 cm between the radiator and your body.

It is designed not to exceed the emission limits for exposure to radio frequency (RF) energy set by the Federal Communications Commission.

This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

1.4 TECHNICAL SUPPORT

If you have any questions about our products, please contact the technical department of Ojmar:

Telephone no.: +34 943 748 484

Fax: +34 943 748 490

Web: www.ojmar.com

2. INSTALLATION MANUAL

2.1 SDK INSTALLATION

OJMAR's API allows you to install both the drivers of the NFC programmer and of the CNReader. The following documents are also copied in the installation process:

- OJM-PR-2015-085 API general overview.pdf
- OJM-PR-2016-118 API reference manual.pdf

2.1.1 Installation

The files you have received and which are necessary to perform the installation are the following:

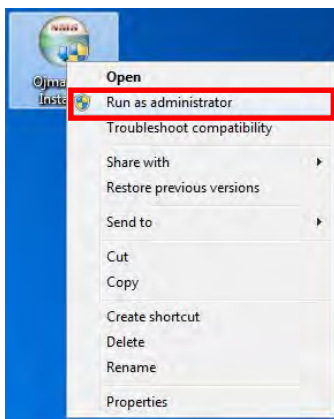
- Installer "Ojmar API Installer.exe".
- Installer licence file.

Follow the steps listed below to carry out the installation:

- **Run the installer:** Run the file "Ojmar API Installer.exe", which will provide the installer of the API.

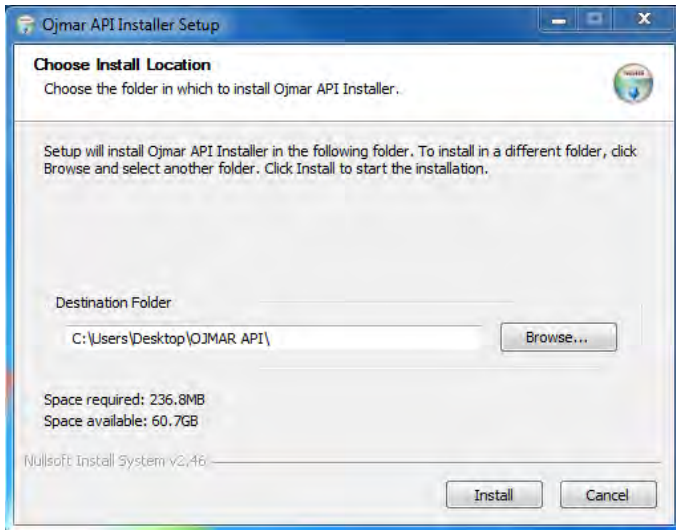


Right click and select "Run as administrator".

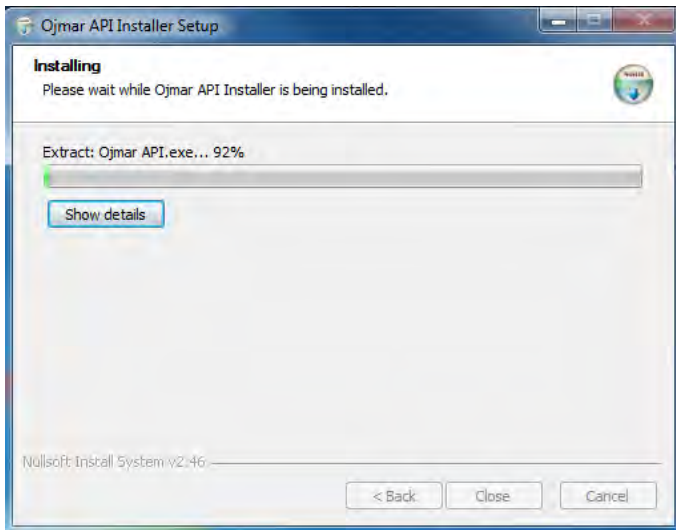


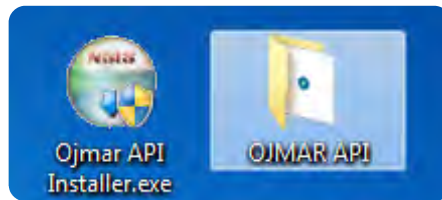
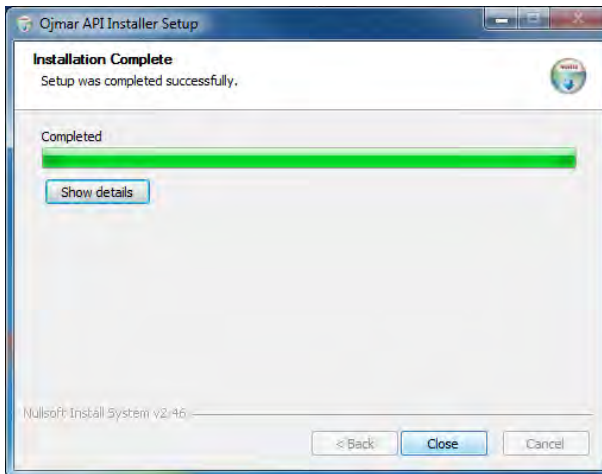
This file decompresses the installer and the jre to a folder.

- **Select the destination folder:** Select the folder where the files will be decompressed.
- Nb: If you did not select a destination folder, the files will be decompressed in the same folder as the source file.



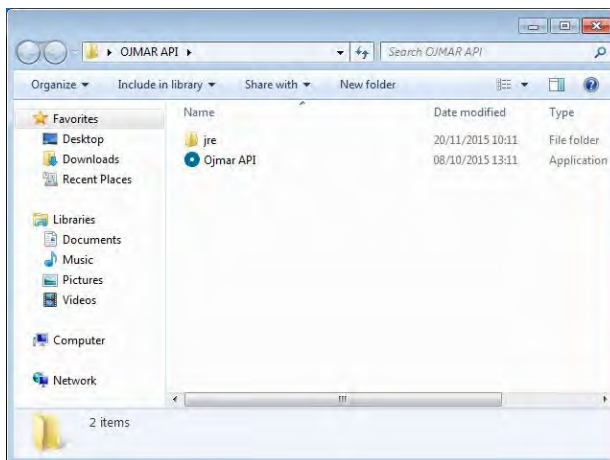
Click on "Install". The decompression will begin. Close the window when the installation is complete.





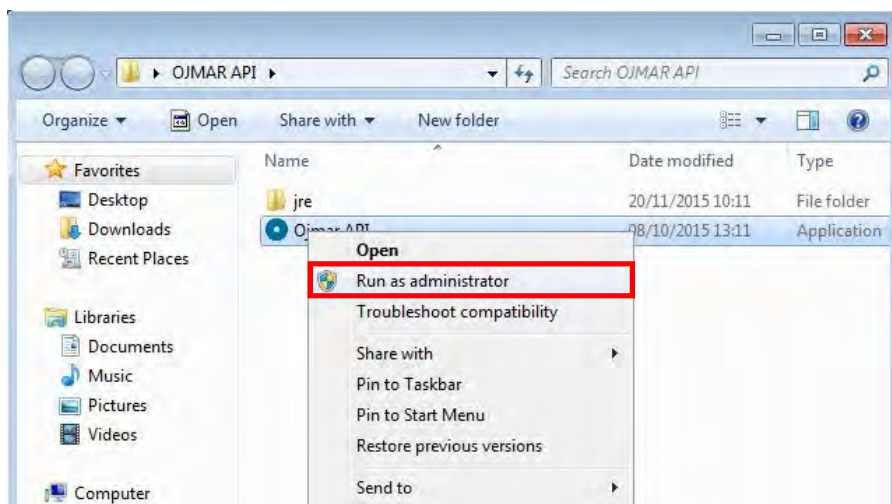
The following files will appear in the selected folder:

- jre
- Ojmar API.exe



2_MANUAL DE INSTALACIÓN

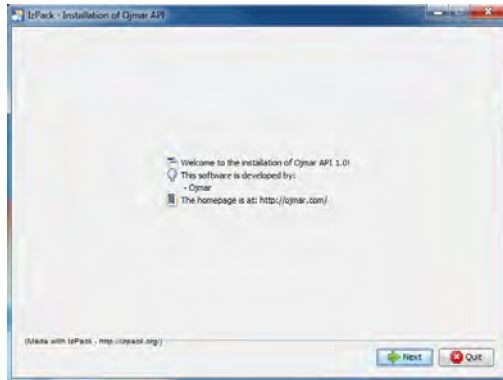
- **Installation of the API:** Run the file “Ojmar API.exe” to start the installation of the API. Right click and select “Run as administrator”.



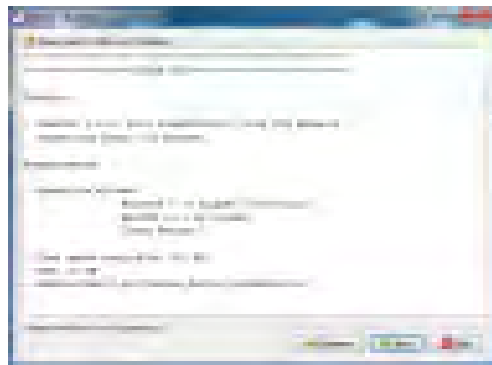
- **Select language:** The first step must be to select the language. The following two languages are available:
 - English
 - Spanish



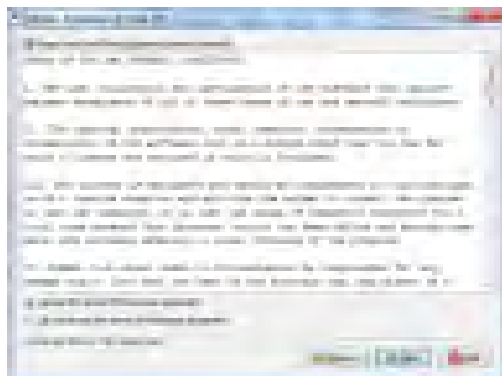
The installation process will begin. Click “Next” to continue.



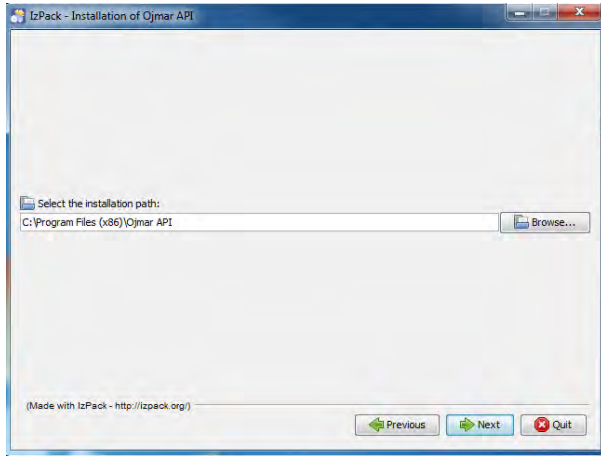
- **Read the information and agree to the terms and conditions:** Press “Next” after reading the available information.



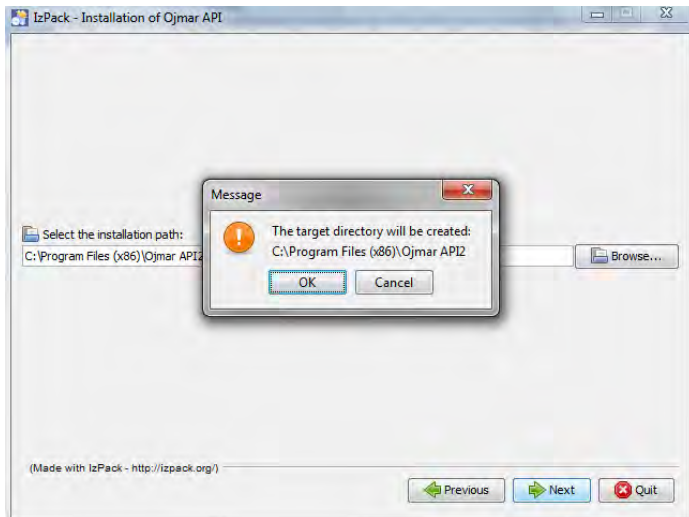
In the next step, you must accept the terms and conditions of the licence agreement to continue.



- **Select the installation folder:** Select the installation folder for the API.



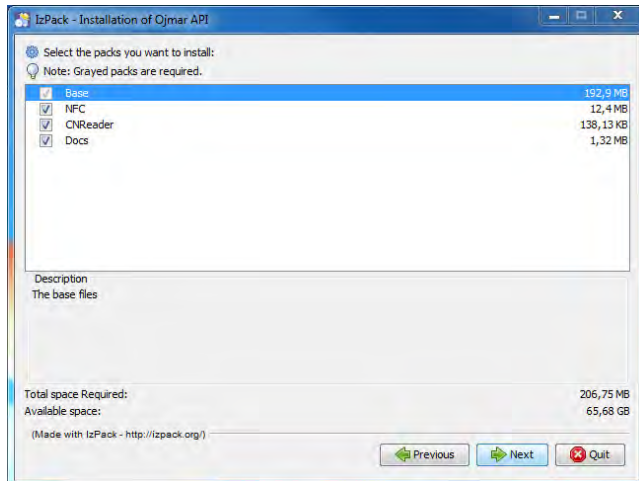
If the folder does not exist, a message will be displayed warning that the folder will be created. Press “Accept” to continue with the process.



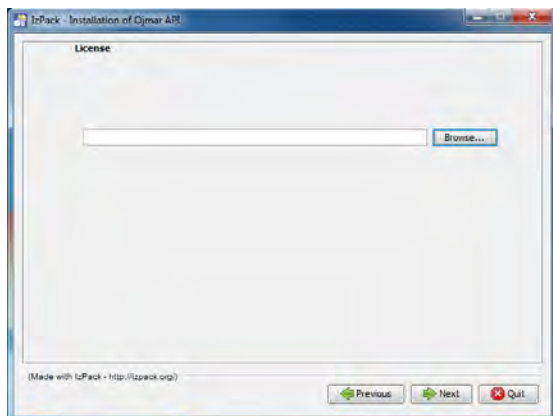
- **Select packages to install:** The API module is required but the installation of the drivers and documentation is optional.

The packages available are as follows:

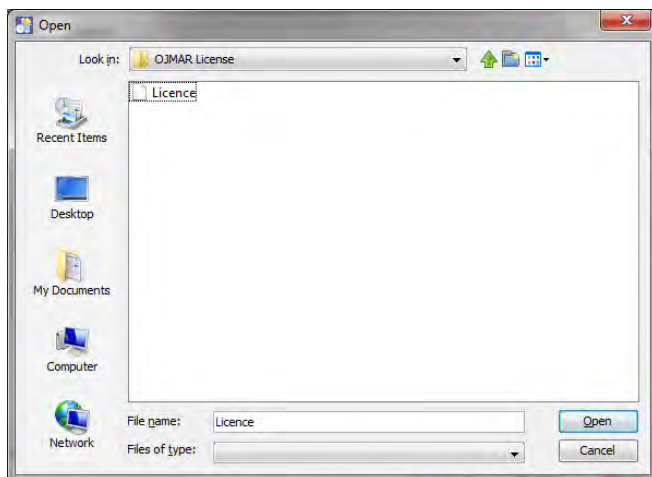
- **Base:** This is the API package. It is not optional.
- **NFC:** Drivers of the NFC programmer. If selected, they will be installed automatically.
- **CNReader:** Desktop reader drivers. If selected, they will be copied to the API folder. Follow the steps of the “**Drivers CNReader**” section to install them.
- **Docs:** Manuals. If selected, they will be copied to the API folder.



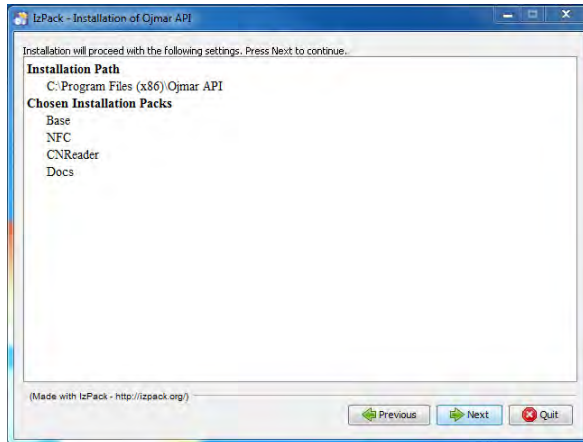
- **Licence:** You must select the licence file.



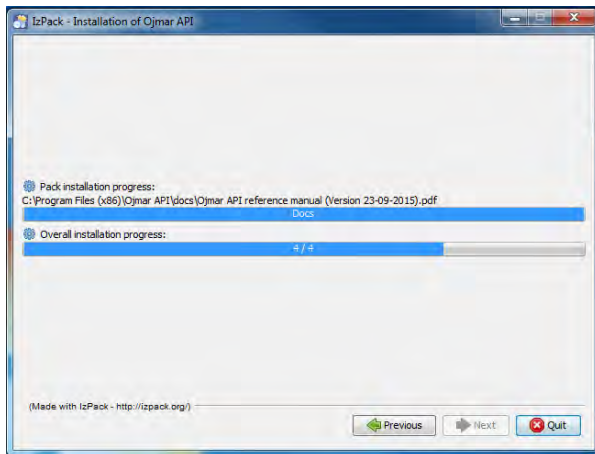
Search for the file in the folder that contains it and press "Next".



- **Confirm installation:** When you press “Next” the installation route and the packages that will be installed will be displayed. Press “Next” to begin the installation.

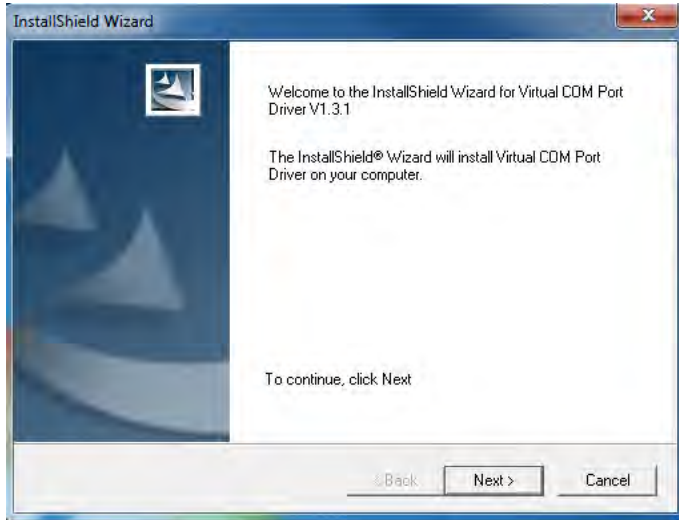


- **Installation process:** The installation progress will be displayed in a window.

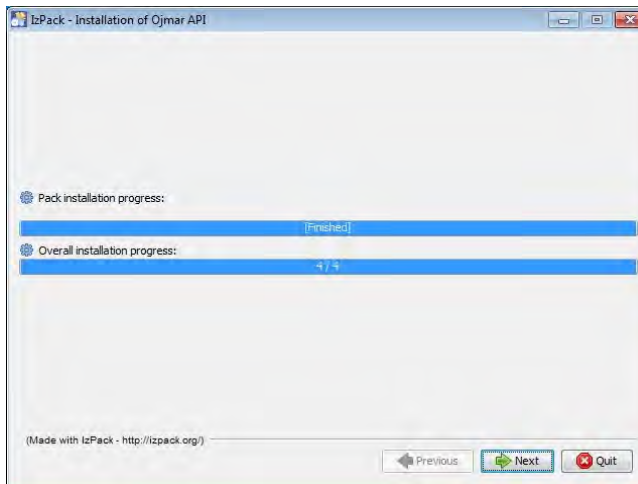


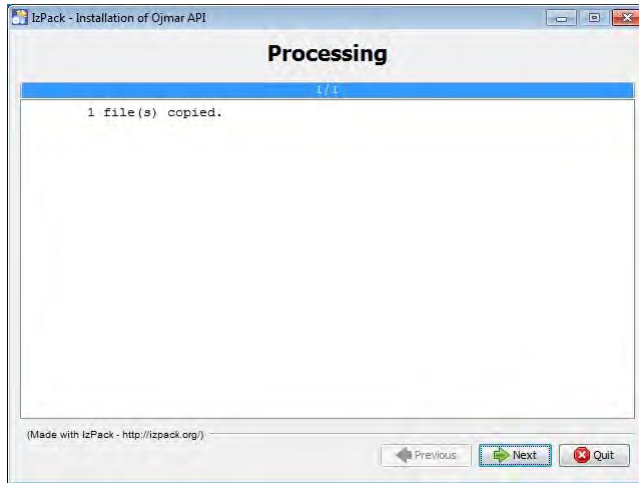
Before finishing, the installation process of the drivers of the NFC programmer will be launched if it has been previously selected.

- **NFC Drivers:** An installation wizard will help you install the drivers.

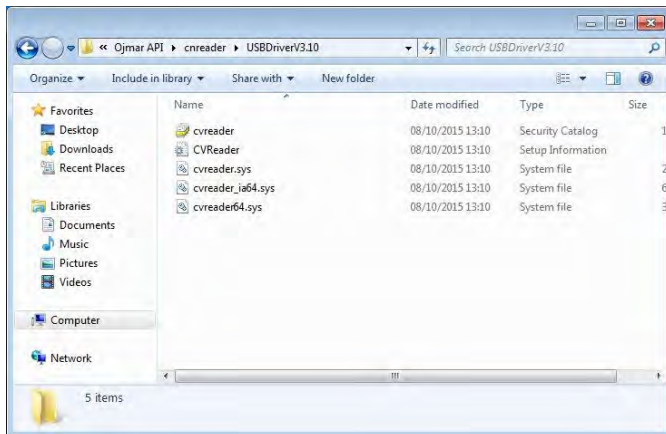


- **Complete the installation:** When the progress of installation has finished, press "Next" to install the shortcuts and complete the process.





- **Drivers CNReader:** The desktop reader drivers are not installed automatically. These are copied to folder \cnreader\USBDriverV3.10 of the folder where the API has been installed.



The supplied drivers are supported for the following operating systems:

- Windows XP (32 and 64 bits)
- Windows Vista (32 and 64 bits)
- Windows 7 (32 and 64 bits)
- Windows 8 (32 and 64 bits)
- Windows 10 (32 and 64 bits)
-

2_MANUAL DE INSTALACIÓN

- NB.: All screens shown below are for Windows 7. If you have another Operating System they may be slightly different.

The procedure to be followed to install the controllers is as follows:

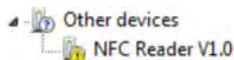
1. Connect the programmer to the USB port of the computer.
2. Windows will detect the programmer and displays the following messages:



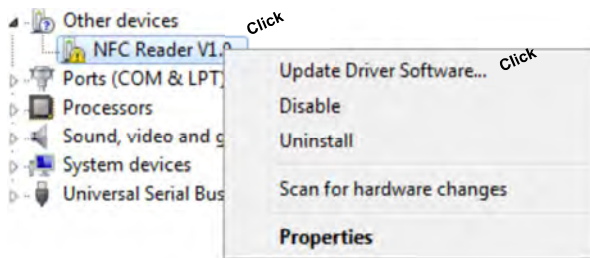
3. Windows will not be able to automatically install the Windows driver, and therefore you will have to install it manually.
4. To do this, access the “Device Manager” of Windows. This option is found in the “Control Panel”.



5. Locate the following error in the list:



6. Right click on the device in conflict and select “Upgrade driver software”:



7. Select “Browse my computer for driver software”.

How do you want to search for driver software?

➔ Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

➔ Browse my computer for driver software
Locate and install driver software manually.

Click

8. Press "Browse" and locate the "Drivers Cnreader" folder of the CD supplied by Ojmar.

Browse for driver software on your computer

Search for driver software in this location:

D:\Drivers\Drivers cnreader

Browse...

Click

☒ Include subfolders

➔ Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Next

Click

Cancel

9. Confirm the driver installation clicking on the button "Install".

Would you like to install this device software?

Name: Civintec
Publisher: CiviTec Global Co., Limited

☐ Always trust software from "CiviTec Global Co., Limited".

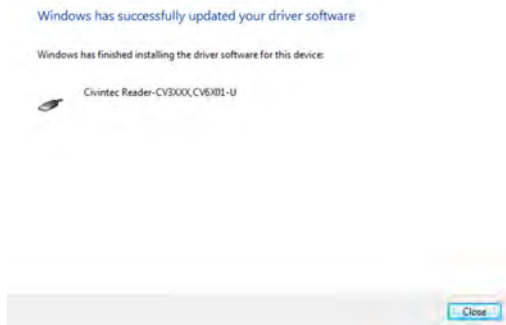
Install

Click

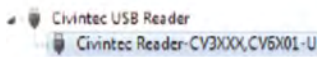
Don't Install

 You should only install driver software from publishers you trust. [How can I decide which device software is safe to install?](#)

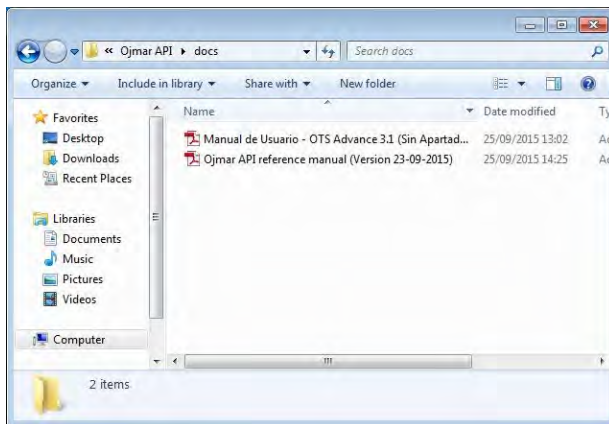
10. If the installation was successful, the following screen will be displayed.



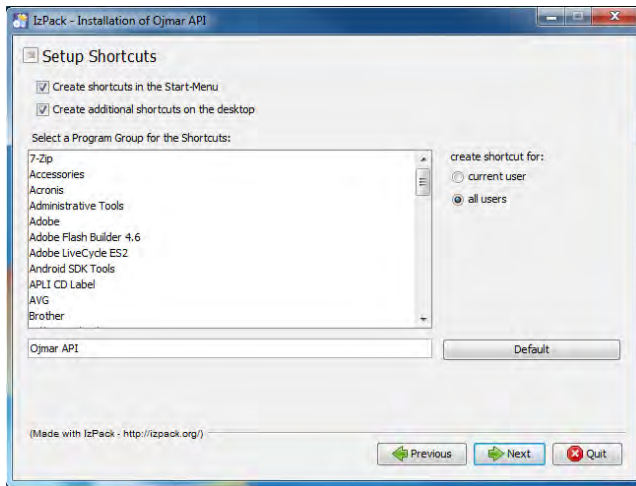
- NB.: If in Device Manager the following picture is displayed, it means that the driver was installed correctly.



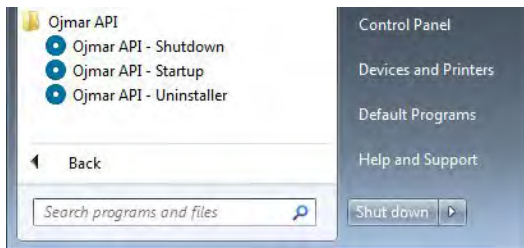
- **Docs:** The manuals are copied in folder \docs of the folder where the API has been installed.



- **Shortcuts:** In the last step, you can create shortcuts to the application. There are two options available:
 - Shortcuts on the desktop.
 - Shortcuts in the start menu.



From the start menu, you can start the application or uninstall it.

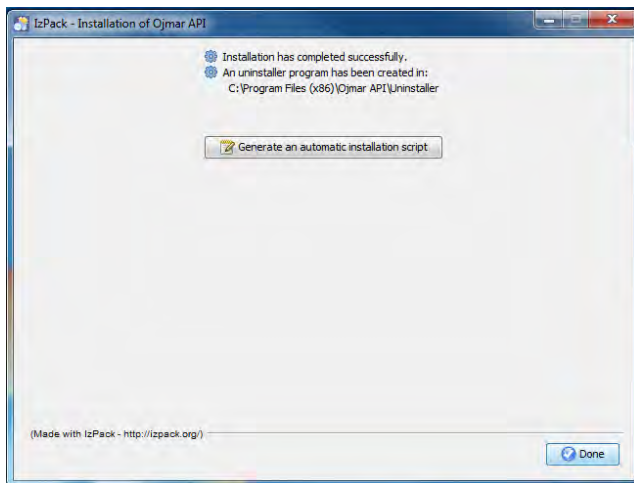


The API can be started by double-clicking on the shortcut on the desktop.



- Run as administrator.

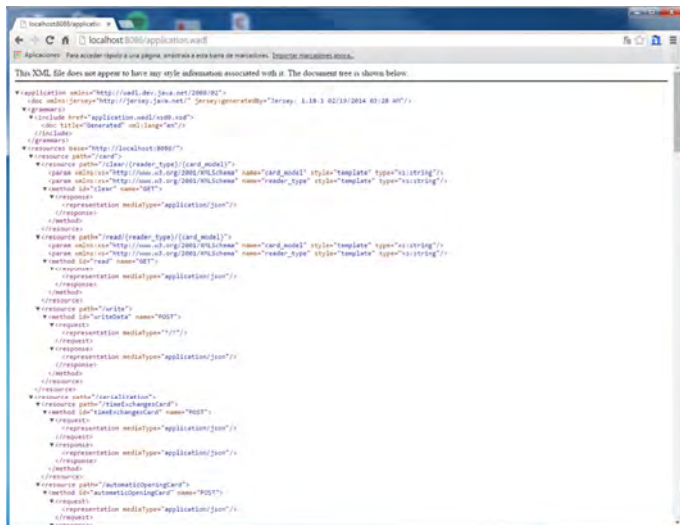
- **Installation complete:** When the installation has been completed, the following message is displayed. Press “Done” to close the window.



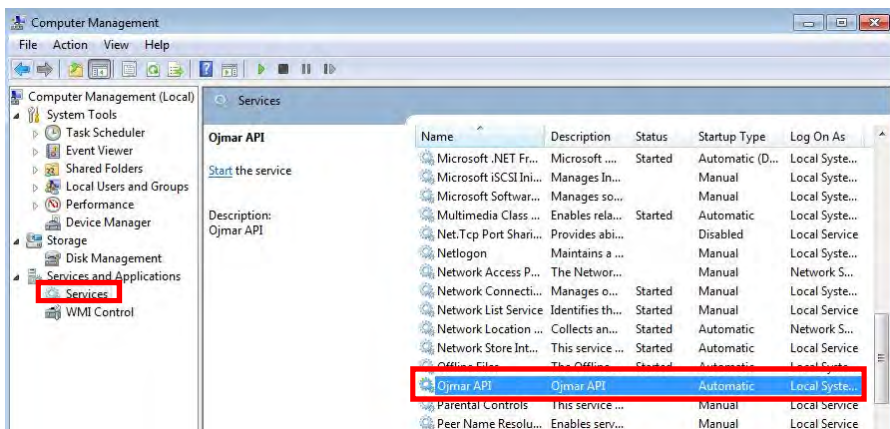
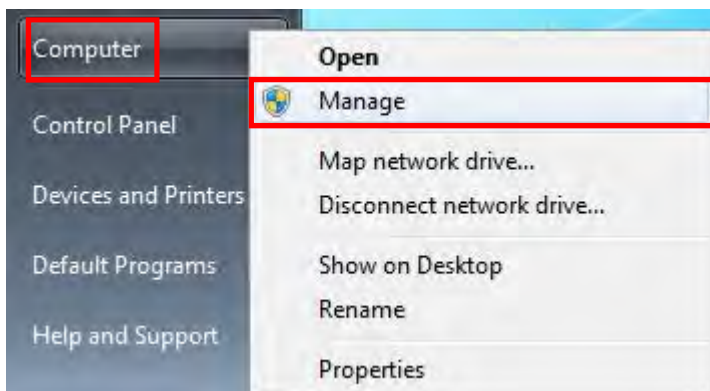
- **Test the API:** You can test whether the API is running by opening the browser and going to the address below.

- <http://localhost:8086/application.wadl>

If the following xml is displayed it means that the API is running properly.



- **Windows Service:** The API is installed as a Windows service. Right click and select “Run as administrator”.



2.2 PHYSICAL INSTALLATION OF THE ROUTER

Configure the router following the manufacturer's specifications in the corresponding user manual. For the correct installation of the router, see section 8.1.

2.3 ASSEMBLY OF THE NEXO NLX1 LOCK

The NEXO NLX1 lock can be installed on doors with thicknesses between 9 mm and 20 mm.

The assembly of the lock on the locker door must be correctly carried out to ensure the correct operation of the lock.

The position of the mounting of the holes, the distance of the lock in the body of the locker and the distance of the lock to the strike, as shown in Figure 2-2 and Figure 2-3, must be carried out correctly to avoid problems during closing.

2.3.1 Lock Direction

Lock installation must be carried out with screws in four drilled holes.

Lock assembly depending on if it is right handed or left handed is as follows:

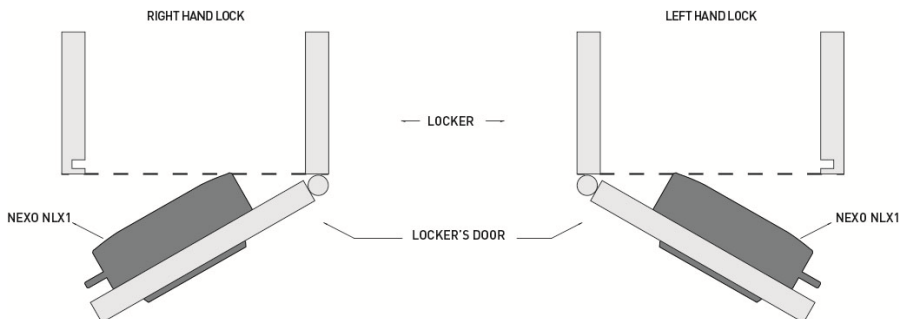


Figure 2-1. Representation of right and left handed locks.

- Nb: It is important for the customer to send a 3D file of the locker (parasolid, step) for Ojmar's technical department to assemble the lock with the strike and ensure that there are no interference problems between the different elements of the locker.

2.3.2 Assembly Considerations

The NEXO NLX1 lock is fixed to the locker door with four screws.

- Recommended screws:
 - Conglomerate doors: Countersunk coach nut 4.5 x 35 DIN 7996 stainless steel (depending on the thickness of the door).
 - Phenolic or metal doors: Countersunk screw M5 x 25 DIN 7985 stainless steel (depending on the thickness of the door).
- The maximum tightening torque to apply to the tightening of each screw or coach nut will be 150 Ncm.
- The diameter of the central hole must be 23 mm (+1/0 mm). Bevelling the hole on the inner area of the door is recommended.

ATTENTION: During installation, ensure that the lock does not collide with the locker body (see Figure 2-2 and Figure 2-3).

The dimensions and location of the anchor points are shown in the following figures (the dimensions depend on the design of the locker and the type of strike used with the lock).

2.3.2.1 Strike with reinforcement

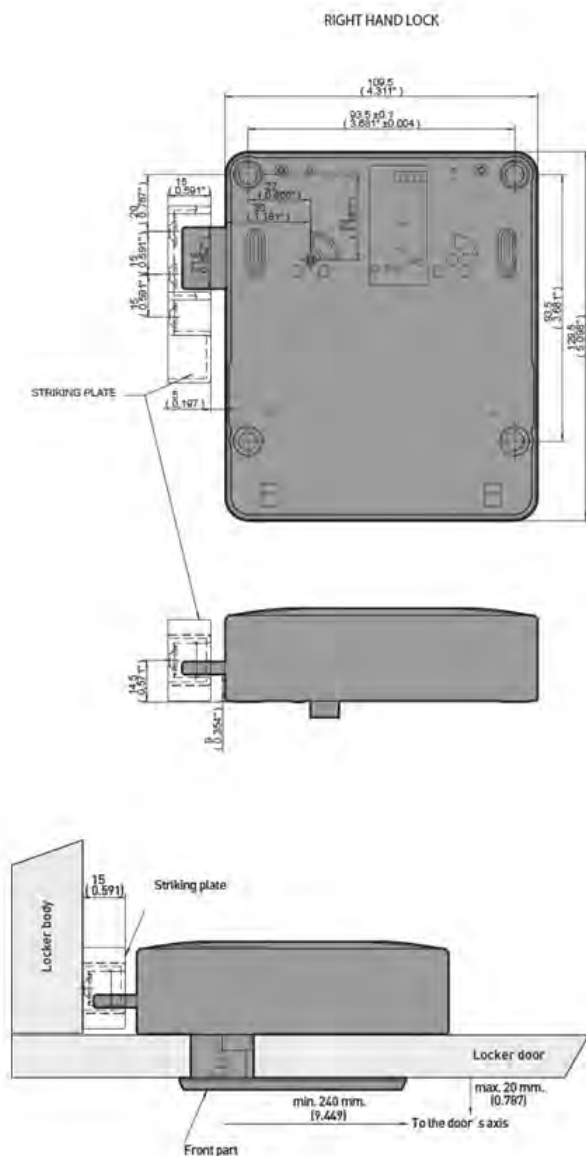


Figure 2-2. Measurements of the strike with reinforcement in mm (inches) (Figure 1 of 2).

LEFT HAND LOCK

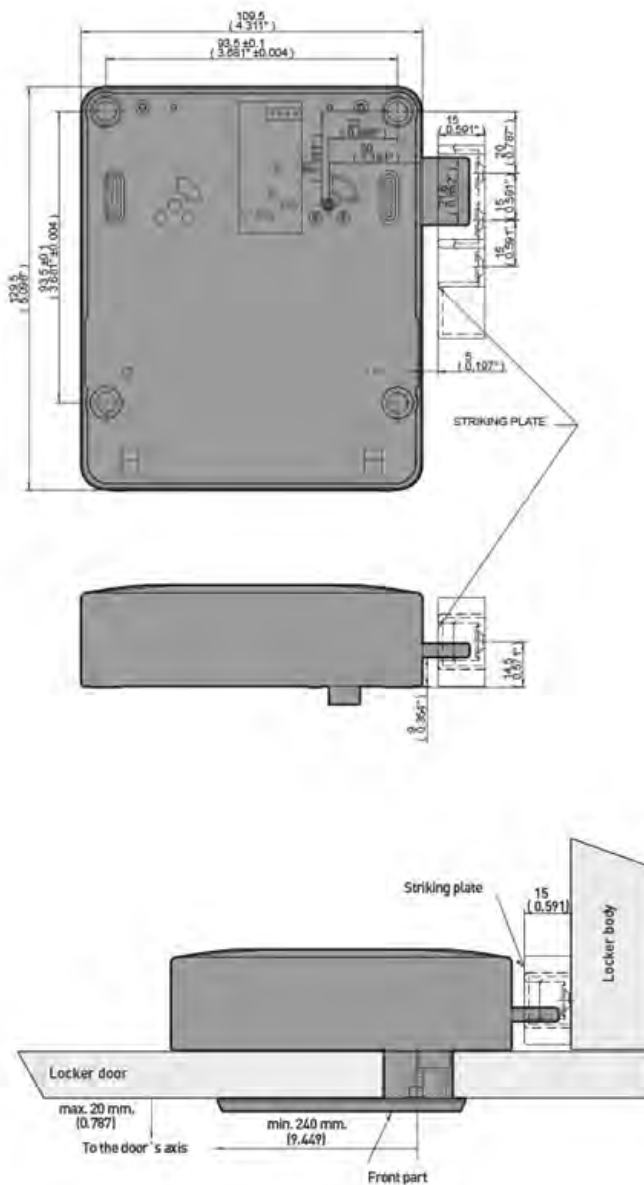


Figure 2-2. Measurements of the strike with reinforcement in mm (inches) (Figure 2 of 2).

2.3.2.2 Smooth strike (with channel in the body of the locker)

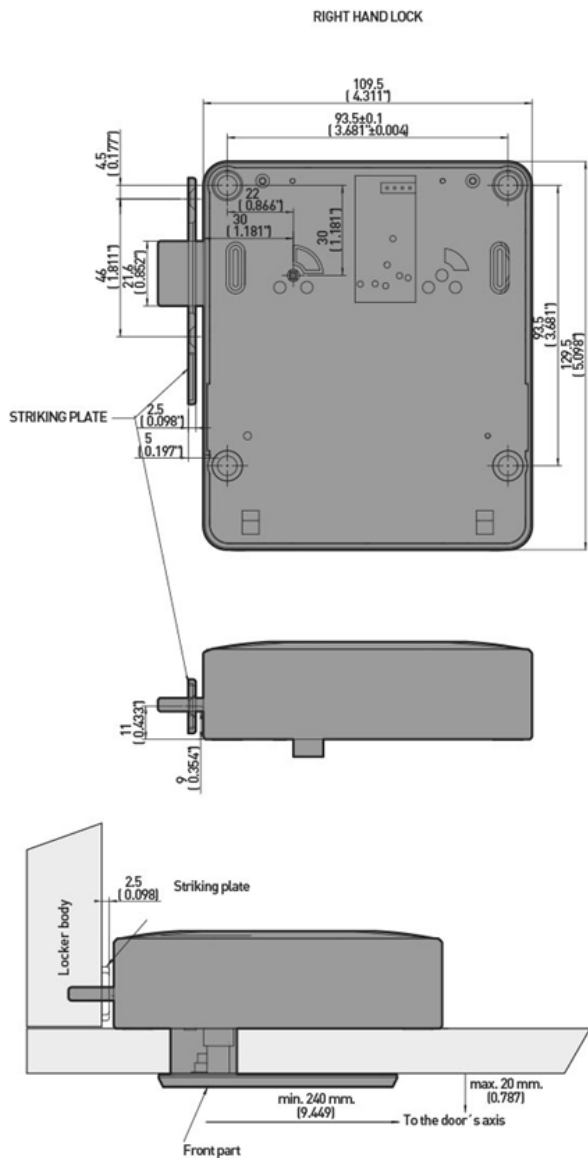


Figure 2-3. Measurements of the smooth strike in mm (inches) (Figure 1 of 2).

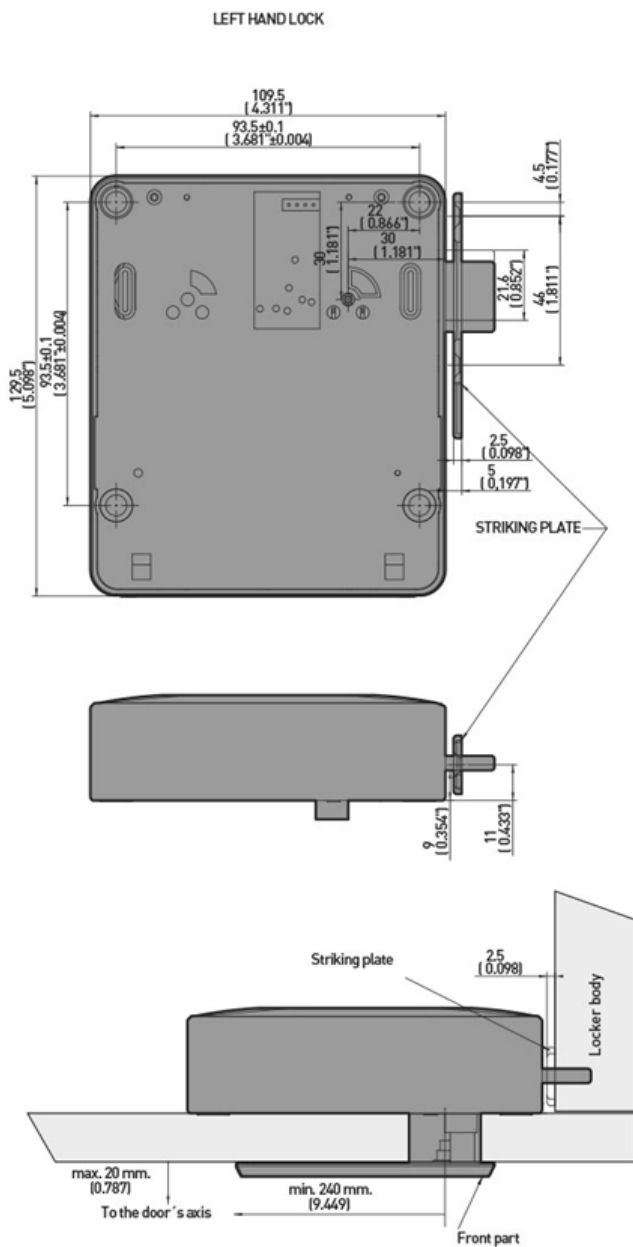


Figure 2-3. Measurements of the smooth strike in mm (inches) (Figure 2 of 2).

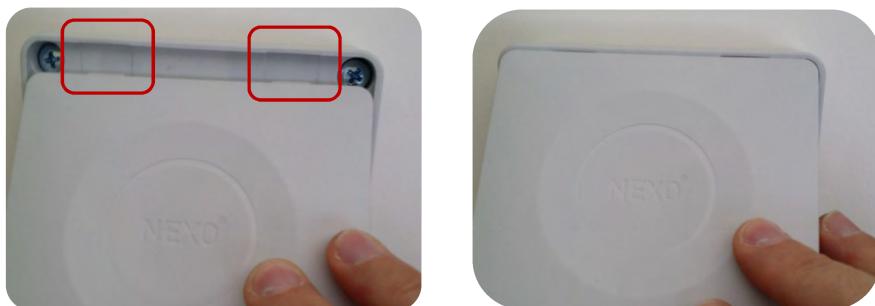
2.3.3 Steps to Follow for Assembly

To assemble a NEXO NLX1 lock on a support or furnishing follow steps in this order:

1. Mark the holes according to Figure 2-4.
2. Drill the central hole of Ø23 (+0.1/0).
3. If necessary, drill the holes to fasten the lock.
4. Screw the lock with four fixing screws. **Ensure that the plastic strip is visible after fixing the lock.**



5. Assemble the rear cover:
 - a. Insert the two tabs of the cover at the top of the lock.



- b. Press the cover at the bottom for clipping the cover onto the lock.



6. Assembly of the led viewer:

- a. Clean the surface of the door and make sure it is dry.
- b. Peel off the plastic film from the emblem.



- c. Insert front panel bulge in the hole connecting of the lock and make sure that the led viewer is aligned with the locker door.



- d. Press the front panel firmly so that it is correctly stuck in the door.



- e. Remove the protective film from the front panel.

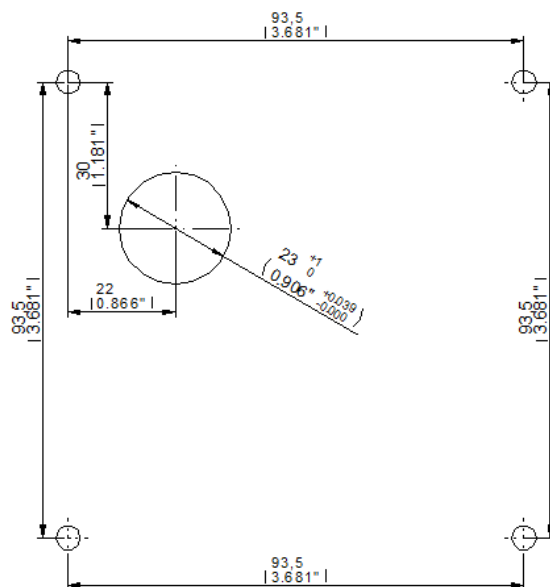


- 7. Remove plastic strip from the bottom side of the lock



8. Fastening the locking device onto the side of the locker. Depending on the locking device to be used, check the measurements shown in designs 1.2 and 1.3. Checking the correct position of the locking device:
 - a. Once the locking device has been tightened, check that the lock can be opened and closed correctly with a user card. Once the locker door is closed, check that the mechanical lock error alarm is not triggered.

POSITION RIGHT LOCK MOUNTING HOLES



POSITION LEFT LOCK MOUNTING HOLES

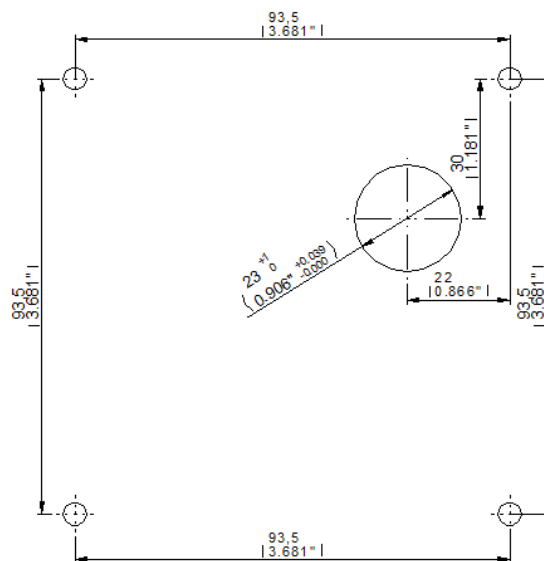


Figure 2-4. Measurements in mm (inches).

PAGE LEFT BLANK INTENTIONALLY

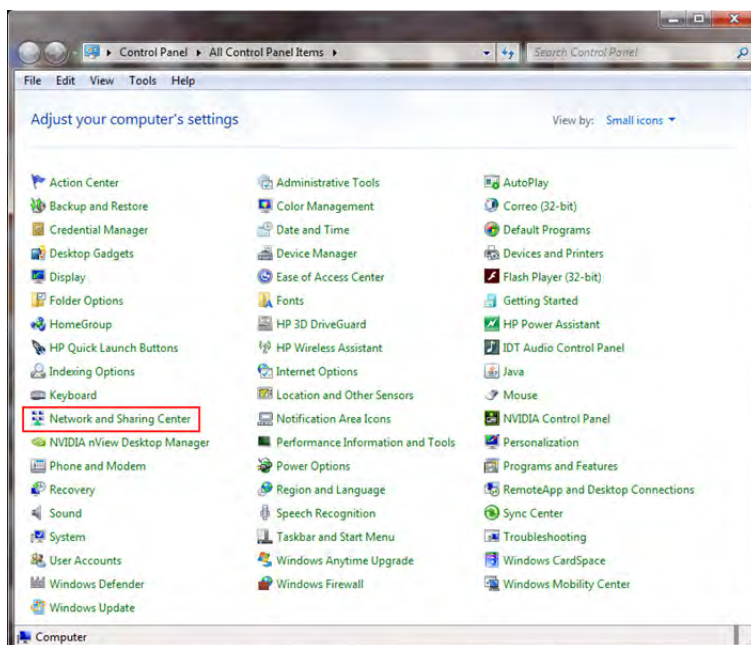
3. START-UP

For more information, see Section 6.6.

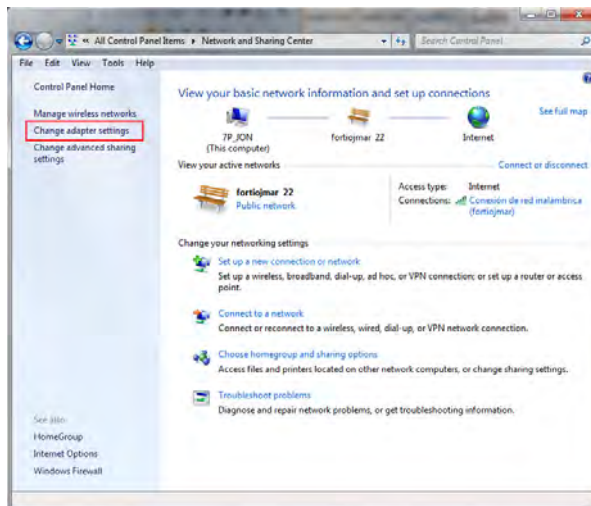
3.1 CENTRAL PC AND PCS (IPS, NETWORK, ETC.)

The PCs connected to the system must have a unique IP address configured, and to do this we need to modify the parameters of the network:

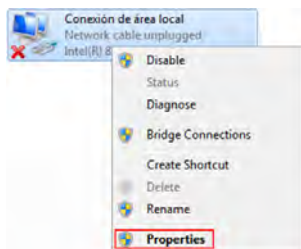
1. Access the network and sharing center in the computer's control panel.
 - The images shown below correspond to the Windows 7 operating system. These images will vary, depending on the installed operating system.



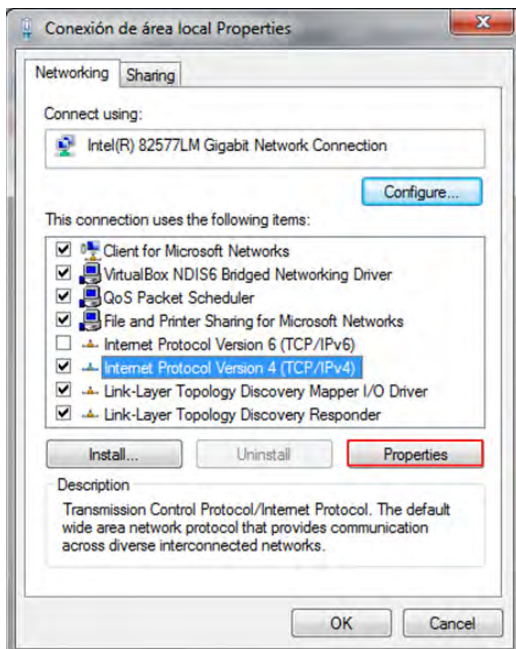
2. Press “Change adapter settings”.



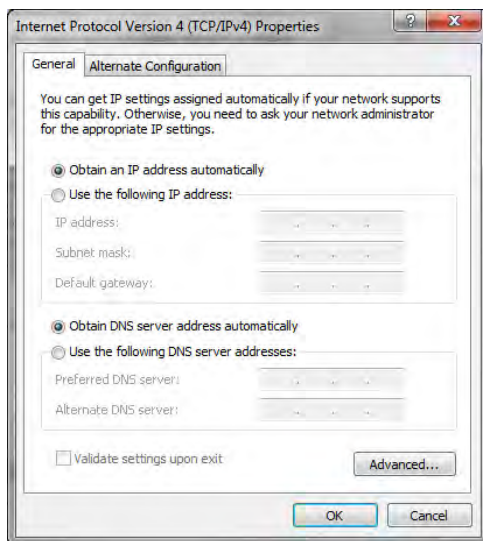
3. Right click the network card and select “Properties”.



4. Select "Internet Protocol version 4", and press properties.



5. You must then complete the following information and click OK:



- To know the range of IPs the router is using, go to “Start/run”, type “cmd” and press enter. Then type “ipconfig”.
- IP Address: IP address you want to assign to the device. To assign IPs statically, you must not repeat an IP in two different devices (locks, router, PCs, mobile phones, tablets, etc.)
- Subnetwork mask: By default, 255.255.255.0. If you want to manage more than 255 devices, change the subnet mask.
- Gateway: IP address of the router.

3.2 READER/PORTABLE PROGRAMMER

Portable programmers allow reading and writing the keys supplied by Ojmar.



NFC OJMAR PROGRAMMER



DESKTOP READER

We have two programmers, each with different characteristics. Both models are connected to the PC via the USB port. The characteristics are detailed below:

3.2.1 Ojmar NFC Model Programmer

The Ojmar NFC programmer works using the installation's Management Software. When working using the Management Software, you can:

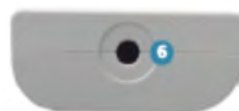
- Read information from the cards/wristbands
- Record user cards/wristbands, cancellation and maintenance cards .
- Initialize locks that have previously been configured from the Software.
- Upgrade the Firmware of the locks.
- NB.: Each time an operation is performed between the NFC programmer and the lock, the programmer updates the time of the lock.



FRONT VIEW



RIGHT SIDE VIEW



FRONT VIEW

1. Touch screen.
 2. Card/wristband reading area: Must be placed in this area so that they can be read/recorded by the programmer.
 3. Screen On/Off:
 - Turn on: Press the button briefly (you will hear a beep).
 - Turn off: Hold button down for 4 seconds.
 4. Connection to PC: USB socket used to connect the programmer to the computer and in this way, load the initialisation data in the programmer.
- NB.: A USB cable is supplied with the reader.

3_PUESTA EN MARCHA

5. Jack connection.

NFC reader: Allows communication between the programmer and the lock. To do this, put the reader of the programmer in front of the symbol on the front part as shown in the picture below.



6. Characteristics:

MAIN CHARACTERISTICS	Operation in autonomous mode or with SW	
	Autonomous mode	Write user cards/Read all kinds of Classic Mifare OTS cards 1k/4k (4B or 7B) and technogym
	Mode of use with SW	Write /Read all kinds of Classic Mifare OTS cards 1k/4k (4B or 7B) and Technogym
		Load the desired configuration in the lock
		Update the configuration in the lock
		Synchronisation with the time of the PC
		Upgrade Firmware of the lock via NFC
TECHNICAL SPECIFICATIONS	Power Supply	USB rechargeable batteries (5,000 mAh)
		Charge via the mains: Use battery charger 5v -1A
	Operating temperature range	-10° to +50° (non-condensing)
	Approx. weight	380g
APPROXIMATE DIMENSIONS	Outer (length x width x height)	202 x 89 x 39 mm

3.2.1.1 Main Screen

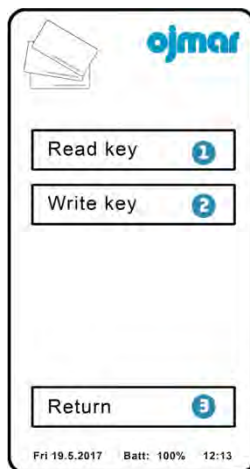
The main screen of the Ojmar NFC programmer displays the following information:



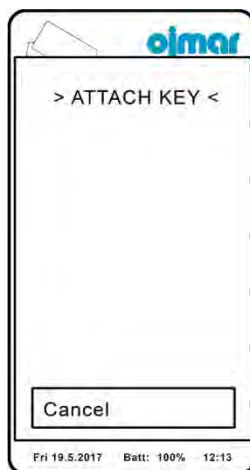
1. Operations with cards/wristbands
2. Operations with locks.
3. Configuration.
4. Date and time: Shows the date and the time of the programmer. This is the information that will be used for recording locks and cards/wristbands.
5. Battery level: Displays programmer's battery level.
 - NB.: The level of the battery is shown for a few seconds after turning on the screen.

3.2.1.2 Operations with Keys

This screen displays the following buttons:



1. Read card/wristband: Read the information associated with the key. Pressing this button displays the following screen:

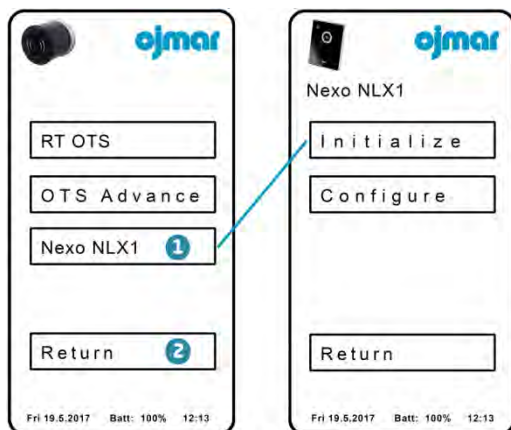


When the card/wristband is put on the reading area the programmer the information associated with this will be displayed.

2. Write card/wristband: Records on the card/wristband the configuration recorded in the programmer. By default, a key of "Free" type will be recorded for 1 lock and in Subgroup 0.
3. Return: Return to the main menu.

3.2.1.3 Operations with Locks

This screen displays the following menus:

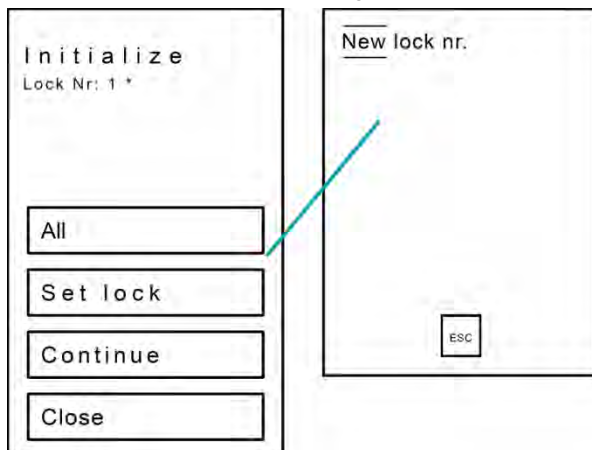


1. Displays the following submenu for the locks of type NEXO NLX1:

- Initialize: This menu option allows initialising locks.
- Configure: This menu option allows configuring locks.

To do so:

- Press “Initialize”.
- Press “Set lock”.
- Select a lock previously loaded in the NFC programmer.
- Press “Continue” and connect the programmer to the lock.

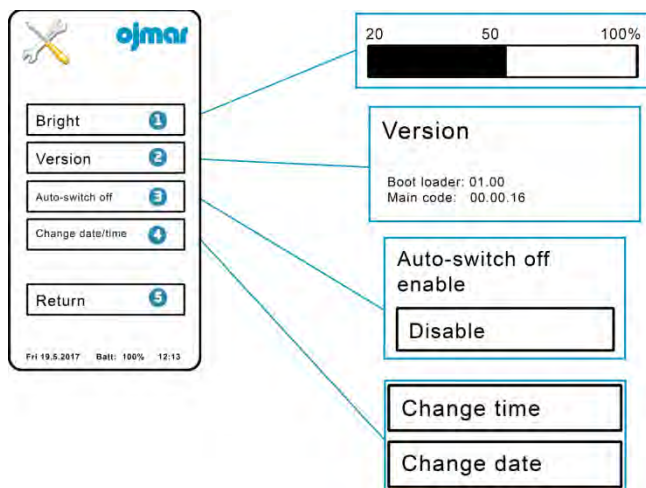


2. Return: Return to the main menu.

3_PUESTA EN MARCHA

3.2.1.4 Settings

This screen displays the following buttons:



1. Bright: Allows regulating the brightness level of the screen.
2. Version: Shows the software version number currently installed in the programmer.
3. Auto power off: When you turn this button on the programmer screen will turn off automatically after approximately 2 minutes of inactivity.
4. Change data/: Allows changing the time and the date of the programmer.
5. Return: Return to the main menu.

3.2.2 Desktop Reader

The Desktop Reader allows reading and recording keys using the installation's Management Software.



- NB.: The desktop reader cannot connect directly with the locks. It must be connected to the computer via the USB port and use the Management Software of the installation for programming the keys.

CHARACTERISTICS

MAIN CHARACTERISTICS	Operation with PC only	
	Write /Read all kinds of Classic Mifare OTS cards 1k/4k (4B or 7B) and Technogym	
TECHNICAL SPECIFICATIONS	Power Supply	Powered by PC via USB
	Operating temperature range	-10° to +50° (non-condensing)
	Approx. weight	83g
APPROXIMATE DIMENSIONS	Outer (length x width x height)	116 x 67 x 14 mm

3.3 LOADING INITIALISATION DATA TO THE NFC PROGRAMMER

See section 7.3.1.1 - Initialisation & set up and point “Configure programmer NDM” of section 7.3.2.3.1.

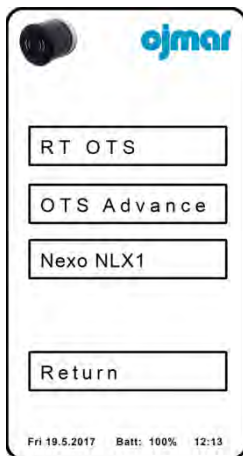
3.4 INITIALISE LOCKS WITH NFC PROGRAMMER

The following steps must be carried out to be able to initialise the locks:

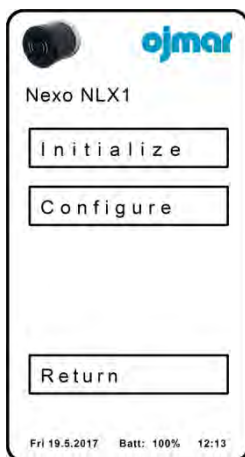
1. Turn on the NFC programmer. Choose option 1 of the next screen.



2. Press button "OTS Online".



3. Press button "Initialize".

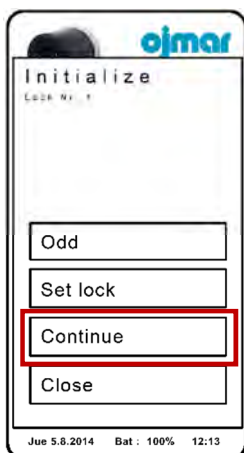


4. Pressing the top button, you can choose between initialising all locks, even or odd locks.

3_PUESTA EN MARCHA



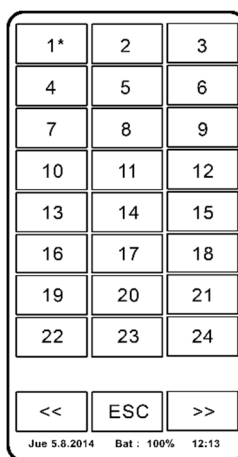
5. Press button "Continue".



6. Hold the programmer in front of the led viewer symbol as shown in the following picture. See the progress in the progress bar of the programmer.



7. A long green led turns on and the lock emits 3 beeps and the green led flashes 2 times, informing that the initialisation data has been recorded correctly.
 8. The PC sends the configuration data using wireless communication to the lock.
 9. The lock beeps 3 times and the green led lights up 3 times, confirming that it has received the configuration data correctly.
 10. The lock is already initialised.
- NB.: By pressing the “Set Lock” button the locks already initialised are displayed, marked with an asterisk.



3.5 DESCRIPTION OF LED FLASHES

NEXUS NLX1 locks incorporate a LED that illuminates whenever an action is performed on it.

The colour codes are listed in the following table:

Colour	Duration flashing 1	Duration flashing 2	Description
GREEN	2x short	2x short	Closed OK.
GREEN	1x short	1x short	Open OK.
GREEN	1x short	No	Reading OK.
RED	20 ms (2.5 s)	No	Locker closed.
RED	1x long	No	Rejected key. Reading Not OK. Impossible to open for whatever reason. Details in NFC programmer and events.
AMBER	3x short	No	Detection low battery.
AMBER	1x long	No	Definitive low battery. It ceases to operate.
GREEN	3x short	3x short	Successful maintenance operation: Initialisation, configuration, fw upgrade.
RED	3x short	3x short	Unsuccessful maintenance operation.
RED	3x short	3x short	Fault. Send to Ojmar: EEPROM deprogrammed, mechanical error...

GREEN: Correct Operation.

AMBER: Pay attention to the message and fix it as soon as possible.

RED: Problem detected.

- Very short: 50 ms
- Short: 300 ms
- Length: 1000 ms

3.5.1 Lists

To manage their opening and closing, the locks of the system have user lists:

- White list: Users who are on this list will be able to operate with the lock in question.
- Black list: Users who are on this list will not be able to operate with the lock in question.

PAGE LEFT BLANK INTENTIONALLY

4. CARD RECORDING

See Section 7.3.2.2.

There is a total of 5 types of different cards.

4.1 FREE

The “Free” operating mode allows access to the lock of any programmed key of this type.

The operation is as follows:

- A free card allows closing and opening any free lock that is not been occupied at that time.
- From the moment that the lock is occupied, the card cannot be used in any other free lock until the former has been freed.

4.2 FIXED

The “Fixed” operating mode allows assigning to a lock a given associated number.

- EXAMPLE: A gym member reserves a locker and only he/she can use it.

Only fixed cards will have access to the assigned lock.

For each Fixed lock, you can create as many cards as you like and all of them can have access at the same time.

- EXAMPLE: A member who has two fixed keys can close a lock with a key and open the same lock with another key.

4.3 MASTER

Use to be able to open any type of lock. This key only allows you to open the lock. In no event does it allow to closing it.

- NB.: When the lock is open its memory is cleared and is now free for another user.
- NB.: A single master key is supplied (in key fob format) for each installation. If you want an additional master key contact Ojmar.
- NB.: For exclusive use of the facility's staff.
-

4.4 SERVICE

Use to be able to open and close any type of lock. This key does not change the lock. When closed, it will continue to be occupied by the key it previously had assigned.

- NB.: A single service key is supplied (in key fob format) for each installation. If you want an additional service key contact Ojmar.
- NB.: For exclusive use of the facility's staff.

4.5 DELETED

This key allows deleting all the information of the lock and returning it to its factory state. In this way, the NFC programmer must be used to initialise the lock and start it up again.

- NB.: A single deletion key is supplied for each installation. If you want an additional deletion key contact Ojmar.
- NB.: For exclusive use of the facility's staff.

5. MAINTENANCE MANUAL

5.1 OPENING OF THE NEXO NLX₁ LOCK USING EMERGENCY POWER SUPPLY

It is advisable to carry out maintenance of the doors of the lockers every 6 months to make sure they are in their correct position and thus, correct any misalignments produced by the hinges and that the closing of the locker is correct.

If case no maintenance has been done and the condition of the batteries has not been checked, the locker may be in the closed position and the lock has run out of batteries. By bringing the key or master or service key close the lock does not respond.

If this happens, you should proceed as follows:

1. Remove the frontal display from the locker. To do this, use a flat-tip screwdriver and detach, taking care not to scratch the door of the locker. Insert the screwdriver between the door of the locker on the lower corner of the viewer and push and lever the door against the door until the frontal display starts to come free.

1



2



3

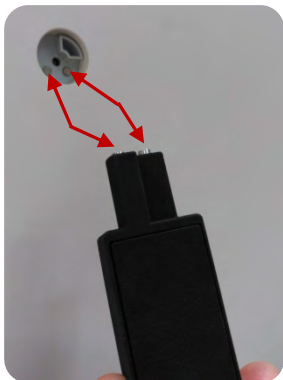


4



2. After unsticking the frontal display, put the emergency power supply tool in the hole of the door until the two pins make contact with lock's pins . This way we power the lock and with a user key, master key or service key, placing it under the hole in the door, we open the locker.
 - Nb: The power supply tool can only be inserted into the hole in one position.

1



2



3



4



User key, master or service

5.2 EMERGENCY MECHANICAL OPENING OF THE NEXO NLX₁ LOCK

If there is a mechanical fault in the lock and is in “closed” position preventing the opening of the locker, or we have carried out the steps in Section 5.1 and it still cannot be opened, proceed as follows:

1. Remove frontal display display. See point 1 of Section 5.1.
2. Once removed, pick up the *emergency mechanical opening tool*, arrows facing up and position it in the door's hole, resting it against the door of the locker as shown in the picture. Insert the diameter of the mechanical opening tool into the hole in the door.

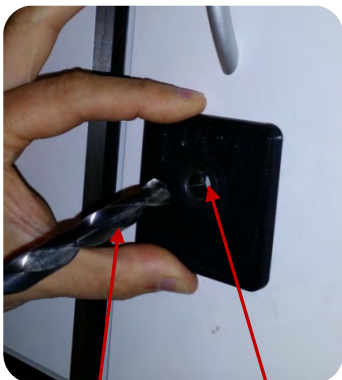


3. Make sure that the *emergency opening tool* is aligned with the door, not in an angle. With a drill and $\varnothing 10$ mm bit, insert the bit into the hole of $\varnothing 10$ mm and drill to make a hole in the lock.

ALIGNMENT OK



ALIGNMENT NOT



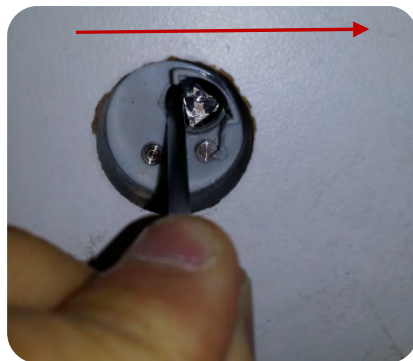
Bit $\varnothing 10$ mm

Hole $\varnothing 10$ mm



4. Insert the key supplied by Ojmar in front of the metal part of the lock and with the help of this, remove it by pushing it to the right if it is a right lock or to the left if it is a left lock.

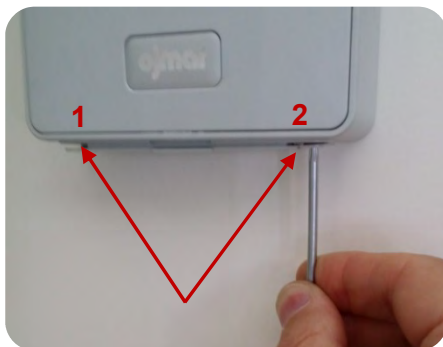
The internal mechanism of the lock will move releasing the locker door.



5.3 DISASSEMBLY OF THE NEXO NLX1 LOCK

The steps to follow to disassemble a NEXO NLX1 lock on a support or furnishing should be followed in this order:

1. Loosen the two screws on the battery holder using the key provided by Ojmar.



2. Remove the battery holder.



3. Remove the cover. To do this, with the flat side of the key supplied by Ojmar, push the clip of the cover upwards that is on the gap on the battery holder.

This clip is located in the central part of the lock.

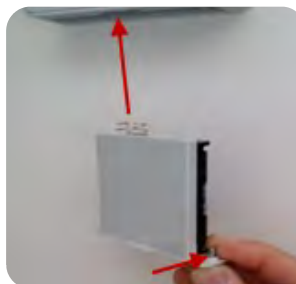
Lower part



4. After unclipping the cover, use the angular area of the key to push the cover and remove it from the lock.



5. Attach the battery holder with the two screws.



6. Loosen the 4 screws of the lock.

5.4 REQUEST FOR DATA: EVENTS, CYCLES, SETUP

See Section 7.3.3.4.2.

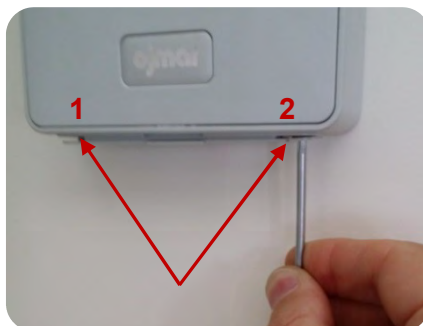
5.5 BATTERY REPLACEMENT

Caution!

- Nb: Explosion risk in case of change to a different type of battery. Move those batteries away according to the instructions given.

The steps to be followed to replace the batteries are as follows:

1. Loosen the two screws on the battery holder using the key provided by Ojmar.



2. Remove the battery holder.

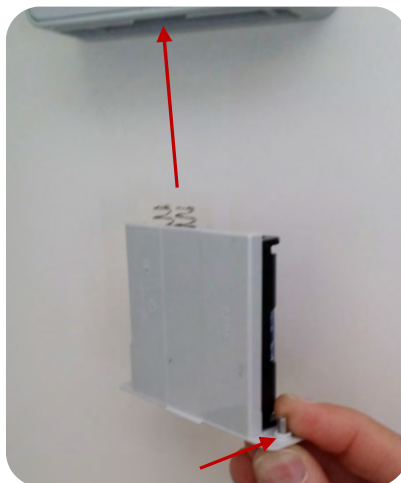


3. Replace the batteries and put the battery holder on again.

The NEXO NLX1 lock needs 4 AA batteries.

- Battery recommended by Ojmar:

VARTA INDUSTRIAL ALKALINE
1,50 V. SIZE AA LR6.



- NB.: The batteries used must be deposited in containers destined exclusively for batteries for their proper management.



5.6 CLEANING

While the lock has been specially designed and produced for continue use in damp environments, it must adhere to the following guidelines for proper maintenance of the same:

- Clean it with a soft, damp cloth (**do not use any detergent product**) and then dry it completely.
- Do not submerge it.
- Protect it from water.
- Do not expose it to direct sunlight or extreme temperatures.
- Do not let it fall.
- Do not subject it to strong blows.
- Do not disassemble it.
- **ATTENTION:** In the event of cleaning the installation with water jets, it is necessary for the doors of the installation's lockers to be closed to keep them free of the effect of corrosive substances that accelerate wear.

5-7 FAQ

QUESTION	CAUSE	SOLUTION
While attempting to initialize with the NFC programmer the lock does not respond.	The programmer is not placed in the correct position.	Put the programmer on the front part as shown in the picture. 
		Reset the lock by releasing the battery holder and replace fastening it with the two screws (wait until the lock is calibrated, the amber led will turn on 7 times).
The lock does not receive the configuration data from the PC.	There is no communication with the router.	Check Ethernet connection.
		Make a ping to the router.
		Make a ping to the PC.
The lock does not open.	The lock is occupied by another key.	Pass the master key (Red) and check that it has released correctly closing and opening the lock with a free user key.
The lock is inoperative.	The lock has detected a definitive low-battery (see flashes lock).	Change the batteries of the lock and check that it is operational by closing it and opening it with a free user key.
The user's card/wristband does not work on the locks.	The card/wristband is occupied in another lock or key is blank.	Check that the card/wristband is blank or occupied reading it in the programmer or the software. If it is occupied, free it using the Management Software or closing and opening the occupied lock. If it is blank, write the card/wristband on the Management Software.

5.8 DECLARATION OF CONFORMITY

I, the undersigned, on behalf of the company:

Ojmar, S.A.

Polígono Industrial de Lerun s/n.

20870 Elgoibar

Spain

Tax Reference A20003042

Hereby declare that the product:

NEXO NLX Lock

Model: 077

Complies with the directives listed below:

- Directive 2004/108/EEC Electromagnetic Compatibility (EMC)
- Directive 1999/5/EC Radio Equipment and Telecommunications Terminal Equipment (RTTE)
- Directive 2011/65/EU Restriction of the use of certain hazardous substances in electrical and electronic equipment (ROHS)

It also declares that the equipment complies with the following European harmonised standards:

- ETSI EN 301 489-1 v1.9.2 (2011-09): "Electromagnetic compatibility and Radio spectrum matters (ERM). Electromagnetic Compatibility (EMC): Standard for radio equipment and services; Part 1: Common technical specifications.
- ETSI EN 301 489-3 v1.6.1 (2013-08): "Electromagnetic compatibility and Radio spectrum matters (ERM). Electromagnetic Compatibility (EMC): Standard for radio equipment and services; Part 3: Specific Conditions for Short Range Devices (SRDS) operating on frequencies between 9 KHz and 40 GHz."
- ETSI EN 301 489-17 v2.2.1 (2012-09): "Electromagnetic compatibility and Radio spectrum matters (ERM). Electromagnetic Compatibility (EMC): Standard for radio equipment; Part 17: Specific conditions for Broadband Data Transmission Systems".
- EN 61000-4-2 (2009): "Electromagnetic compatibility (EMC) - Part 4-2: Electrostatic Discharge Immunity Test".
- EN 61000-4-3 (2006) + A1(2008) + A2(2010): "Electromagnetic compatibility (EMC) - Part 4-3: Testing and measurement techniques - Radiated, radio-frequency, electromagnetic field immunity test".
- ETSI EN 300 328 v1.9.1 (2015-02): "Electromagnetic compatibility and Radio spectrum Matters (ERM);

Wideband transmission systems; Data transmission equipment operating in the 2,4 GHz ISM band and using wide band modulation techniques; Harmonized EN covering the essential requirements of article 3.2 of the R&TTE Directive".

- ETSI EN 302 291 v1.1.1 (2005-07): "Electromagnetic compatibility and Radio spectrum Matters (ERM);

5_MANUAL DE MANTENIMIENTO

Short Range Devices (SRD); Close Range Inductive Data Communication equipment operating at 13,56 MHz; Part 2: Harmonized EN under article 3.2 of the R&TTE Directive".



Ramon Gubikayevskoe
Director gerente de Ojmar, S.A.

Elgoibar 02 de agosto de 2015

Edición: 01

PAGE LEFT BLANK INTENTIONALLY

6. API INTRODUCTION

6.1 PURPOSE OF THE DOCUMENT

The aim of this document is to make an introduction to the API of Ojmar's locking systems to support 3rd party integrators to understand and integrate the API in their system.

6.2 SYSTEM INTRODUCTION

Ojmar's locking system is composed by an API, peripherals to transfer information between the PC and the locks: RFID Cards, a card reader, an NFC programmer, a wireless network and the locks. There will be also a 3rd party software that uses the API. This software is out of the scope of this document. There is also an Ojmar SW available except for the OTS NLX1 lock.

The following peripherals are available in Ojmar's RFID systems:

- NFC Programmer.
- Desktop reader.

And the following type of locks:

- Basic OTS.
- Real time OTS.
- OTS Advance.
- Nexo NLX1 lock.

NFC programmer is able to communicate directly with the lock, the API with the PC application and, can record information into RFID cards.

On the other hand, the desktop reader can record information into RFID cards.



Figure 6-1. Ojmar's RFID system.

Figure above depicts how the system interacts. The API is usually installed in at least one PC at premises and will interact via USB with the NFC programmer and with the desktop reader.

This makes possible to transfer information to the lock by both means the NFC programmer and RFID cards previously recorded by the NFC programmer or the desktop programmer.

6.2.1 NEXO NLX1 Particularities

This lock can work in both online and offline mode simultaneously. When it is in online mode and has access to the server the lock will request to the server access rights to read card, and will operate using the information received from the server. If there is no server access, the lock will decide to provide access rights using the local information contained in the lock (white or black list) and the card. Once the lock performs the action the lock will send the last event of the operation done.

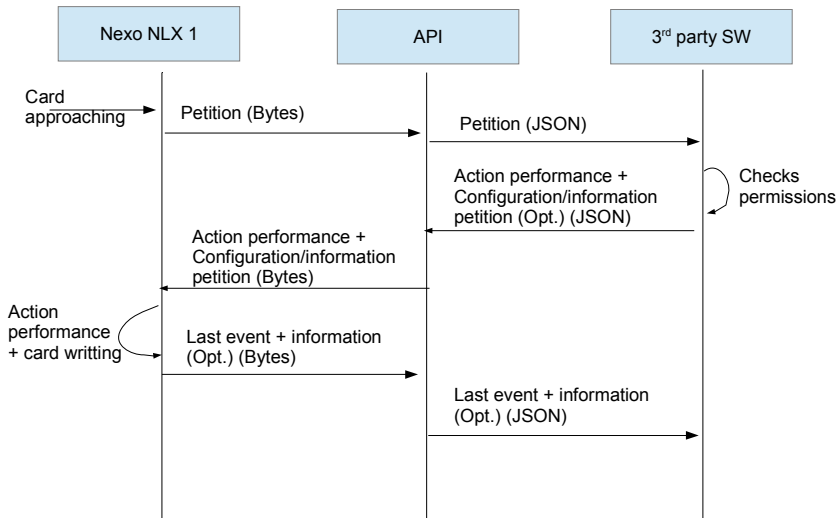


Figure 6-2. System workflow.

To decide if the lock has access or not the 3rd party software should use its own means to keep tracks of the card UUID recorded to each user and the access rights provided to him. The API will act as a mere intermediary in this case.

6.2.1.1 Nexo NLX₁ limitations

Following limitations are defined during the usage of the Nexo NLX1 system.

- Communication limitations:
 - Up to 252 bytes can be sent by TCP/IP communication.
 - Up to 768 bytes can be sent by USB communication.
- Functionalities limitations:
 - Online operations:
 - Up to 36 different profiles can be uploaded per lock
 - Last 10 events can be viewed per communication.
 - Just one information request can be asked per communication.
 - Offline operations:
 - Automatic opening is not available.
 - DST is not available.
 - Time zones and expiry dates for user cards are not available.
 - Up to 125 UUIDs can be uploaded on the lock for white/black list for the offline operation.

6.3 API STRUCTURE OVERVIEW

After API installation, the PC structure will be as follows:

3rd party SW
API Interface
(JSON REST server)
Drivers
Physical peripheral (USB)

More detailed:

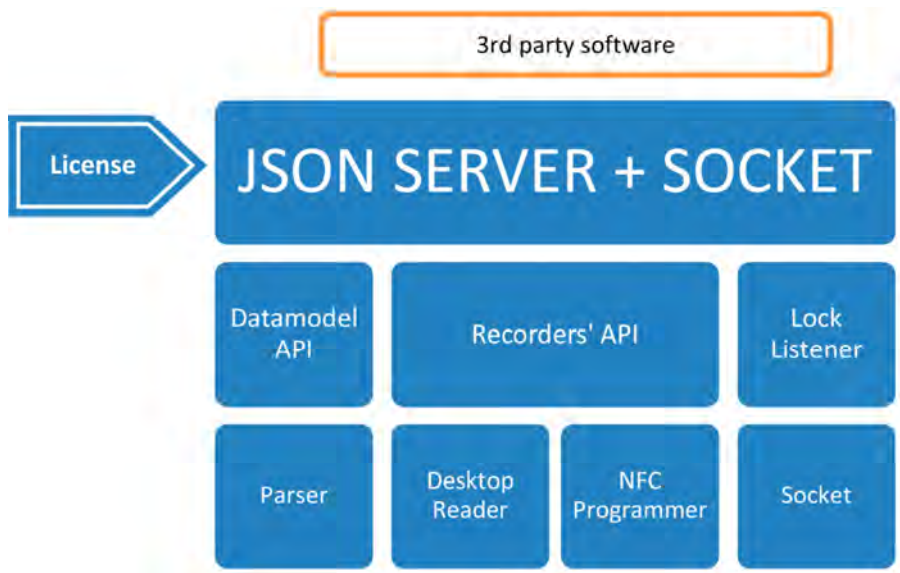


Figure 6-3. API Overview.

The API is implemented in Java so any operating system with an implementation of the Java Virtual Machine may be compatible with the API including Windows, Linux and OS/X.

The API interface is implemented by means of JSON REST Webservices, which warrants a great compatibility with any programming language that includes JSON REST Support. There are many languages that support this; the API documentation has examples for at least PHP, C# and Java.

The 3rd party integrator will receive an API installer from Ojmar and will install it in the PC.

This API includes documentation explaining how to implement communication with the API, and how to implement the functions to be able to make all the system work correctly.

6_API INTRODUCTION

The API needs a valid license provided by Ojmar to work, this license activates the API.

3rd party SW will make REST calls to the API functions via http protocol. The call information will be exchange using a JSON structure (See point examples below in this document).

The API allows to the 3rd party SW to operate in a programmatic way with the locks and the peripherals.

The API is divided in two three parts:

1. **Datamodel API:** Permits the interpretation of the obtained information of the locks and cards independently of the version of lock that operates in the facility. This part of the API works as a parser, so, it is able to encrypt/decrypt the information of the card/lock. These functions are the ones defined in point 3 in the RFID part.

(Communications are encrypted due to security reasons).

2. **Recorders' API:** The API implements the communication with both peripherals, including functions to write and read cards and all the NFC programmer functions.
3. **Socket API:** The API implements a mechanism based on sockets to receive information from the locks and to send answers back to them.

In order to work properly with the API, firstly a frame must be created using the Datamodel API (at the end, it is like a translation from human language to lock language), and after that, it is decided where to send that information (Desktop reader or NFC reader).

This has been implemented in such a way, to avoid reader monopolization, letting to the 3rd party SW to use their own RFID reader writer or the option to use Ojmar's programmer for their own application.

6.3.1 Online Communications

The Nexa NLX1 locks have the ability to work online in a wireless network and communicate with the API. This way the 3rd party SW can receive state information in real time from the locks and determine whether a card should be able to operate with a lock in each situation.

This scenario presents an asynchronous communication where the connection can be started by the lock in any moment.

To make possible this decision delegation the 3rd party SW needs to publish a service for the API to query when a request is received. For that purpose, the API expects a TCP socket to be listening to the machine in a configurable port. The 3rd party SW is responsible to open this socket and implement the protocol to communicate with the online system of the API.

For each connection, the API will forward the information received from the lock translated to JSON format and the 3rd party SW will respond with another JSON.

6.4 DOCUMENTATION PROVIDED

The following documentation is included in the API deliverable package:

- API Introduction. (This document).
- API Reference manual with examples.
- API Installer.
- API License.

6.5 SYSTEMS WORKFLOW

Ojmar's systems workflow is divided in three steps:

1. Device configuration: System initialization, configuration and start-up.
2. Users' configuration: System daily operation with final customer and client.
3. System maintenance: System maintenance operations.

All these operations are performed using RFID cards recorded with the desktop reader or the NFC programmer, or by means of the NFC Programmer itself. The commands to be recorded into the cards or into the NFC programmer are commanded from the API.

6.5.1 Device Configuration

Once the lock devices are mounted, they must be setup for the customers' usage. These devices can be configured for sporadic users, facility members, etc. Following functions can be used for the configuration of these devices.

6.5.1.1 Configuration frames

- Frame for initialization: Sets the basic network, lock number and facility information. Used only for Nexo NLX1 family.
- Frame for time set card: Sets the time and date of the PC in the locking device. This card is mandatory if locks maintenance has to be done.
- Frame for set up card: Configures the lock defining if it is for daily users (free type lock), for facility members (dedicated type locks), to which subgroup will belong to, lock number, etc. (Mandatory). Used only for OTS family.
- Frame for time exchanges card: Defines the time advances and time delays that occur during the year so that the lock updates its internal date automatically. Used only for OTS family. (Optional).
- Automatic opening card: Defines an electronic automatic opening every day so that the lock will be opened every day at the time defined. Used only for OTS family (Optional).

6.5.1.2 Set-up via NFC programmer

Write lock set-up data for OTS family: In this function, the lock data is uploaded to the programmer, alongside with the time and date and the black list. With this

6_API INTRODUCTION

option, more than one different type of lock can be configured in the same step. Used only for the OTS family.

6.5.1.3 Configuration via Socket for Nexo NLX1

Configuration Frame: In order to facilitate the Nexo NLX1 setup the locks may be initialized with basic configuration using the NFC programmer. The lock will connect using the network data and request to the 3rd party SW via API socket the rest of the information (RFID and user configuration).

6.5.2 Customers' usage

Once the devices are configured, cards for the final users can be written. The following function is used for this purpose:

- Frame for user simple card: Creates a frame for a user card that will manage occasional or dedicated locks.
- Frame for user card: Defines which options will be available for that user. Which lock number has access to can be defined, or how many locks can occupy with the same card, which subgroups has option to work with, define an expiry date, time zones, actions, etc.

6.5.3 Devices' maintenance

Now that the installation is running, Ojmar's systems offer the option to recollect different maintenance data so the facility knows how the system is working.

Maintenance, as configuration, can be done via cards or via NFC programmer.

6.5.3.1 Maintenance via RFID cards

- Describe card: Translates a frame with information read from a card to Human readable information.
- Frame for master card: Creates a frame for a card that can open every lock of the facility and erases the lock occupancy if it closed, so that another user can close that lock afterwards.
- Frame for service card: Create a frame for a card that can close and open a lock without deleting any of its memory, so if the lock was occupied, after using the service card the lock is still occupied.
- Frame for cancellation card: Creates a frame where one card can be cancelled and sent to the lock's black list. Used only for the OTS family. If the user card is configured to work with black list is available for NLX1 family.
- Frame for test card: A card that can close and open any non-configured lock. This card is very useful for the mounters, so they can check that mechanical part of the lock fits with the locker. Available for the OTS family.
- Frame for reset card: Send a configured lock to default status (to non-configured lock).
- Frame for subgroup master card: If more than one subgroup is used and different level of maintenances wants to be establish, these card works like a service card but only for the allowed subgroups. Available for the OTS family
- Frame for events card: Creates a frame that can recover the last 3 events of the lock, telling who has made the action, which is the action made to the lock and when has happened that action. Available for the OTS family.

6.5.3.2 Maintenance via NFC programmer

- Write lock set up data: This action can be used over a configured lock in order to update the black list. Available for the OTS family.
- Get events from NFC programmer: Can recover at the same time the last 32 events of 100 locks, telling which action has been made, who has made the action and when has happened. Available for the OTS family

6.5.3.3 Maintenance Via Network

- All the maintenance frames to be recorded in cards depicted in previous paragraphs are also available to be sent to the lock in online mode. Plus: Set Time, Get Events and battery, Set White Black list for offline mode.


```
"card_model_setup":"classic",
"subgroup":2}
```

And the answer shown is:

```
{ "result": "ok",  
  "data":  
    "66660000D80020000000000000000000000000000049000001C7000000000000  
0000000000000000000000000000000000"} }
```

Once received the frame, the information is written to the card in the same way as the set time card. <http://localhost:8086/card/write> with following body:

```
{ "reader_type": "cnreader",  
  "card_model": "desfire",  
  "frame": "  
66660000D80020000000000000000000000000000049000001C7000000000000  
0000000000000000000000000000000000000000"} }
```

And following response with be received, ending the setup card write:

```
{
  "result": "ok",
  "data_ots": "Card write ok",
  "uid": "00000011223344"}

```

6.6.2 Configured by NFC programmer for OTS family

If configuration is made via NFC programmer it can be done in only one step (Second part of the API). The following url should be typed:

http://localhost:8086/nfc/writeInitialization with following body:

```
{
  "locks": [
    {
      "type": "free",
      "number": 53,
      "facilityNumber": "6666",
      "subgroup": 1,
      "config": "",
      "dst_advances": "",
      "dst_delays": ""
    },
    {
      "type": "dedicated",
      "number": 1,
      "facilityNumber": "2222",
      "subgroup": 0,
      "config": "",
      "dst_advances": "",
      "dst_delays": ""
    }
  ],
  "blacklist": ["12345678"]
}
```

And following response will be given:

```
{
  "result": "ok",
  "data": "Init data write ok"
}
```

6.6.3 Set up via NFC for Nexo NLX1 family

First of all, we need to provide the network information to the lock. We will generate an initialization frame to send it to the NFC programmer:

<http://localhost:8086/initialization/initData> with following body:

```
{
  "num_installation": "B1B2",
  "num_subinstallation": "01",
  "tech_type": "classic",
  "data_model_version": "020000",
  "ssid": "OnlineSystem",
  "password": "A1A2A3A4",
  "encrypt_type": "2",
  "server_ip": "192.168.1.97",
  "initial_ip": "192.168.1.100",
  "gateway_ip": "192.168.1.10",
  "range": "1-50",
  "mask": "255.255.255.0",
  "first_num_lock": 1,
  "last_num_lock": 50,
  "group": 0,
  "target": "pp"
}
```

And the following response will be given:

```
{
  "result": "ok",
  "data":
  "00051111B1B201098D020A0400000001000032B1B2030A04000000010000
  32111110409040000000100003201050B0400000001000032000001145B040
  00000010000324F6E6C696E6553797374656D000000000000000000000000
  000000000000000041314132413341340000000000000000000000000000
  00000000000000000002C0A80161C0A80164C0A8010A0132FFFFFF000A03
  010800FE02FDA4"
}
```

6.6.4 Configured via TCP/IP for Nexo NLX₁ family

Once the programmer has the initialization frame, we can place the NFC programmer in a stock factory lock. After that, the lock will get connected to the wireless network and send a request packet to the API at the IP passed in the initialization.

The API receives the request, translates it to JSON and forwards it to the 3rd party SW socket:

```
{
    "profiles":[],
    "uid":"000000D4817BAA",
    "mem_uid":"0000000000000000",
    "lock_status":["A","L"],
    "num_installation":"B1B2",
    "num_subinstallation":"01",
    "locks":[],
    "lock_request":true,
    "lock_response":false,
    "num_lock":1
}
```

The SW will check that the lock number 1 has pending configuration and will respond with the config flag enabled:

```
{
    "num_lock": 1,
    "config": true,
    "num_installation": "B1B2",
    "num_subinstallation": "01",
    "dedicated_locks": [],
    "free_locks": 3,
    "profiles": []
}
```

The lock has been configured.

7. API REFERENCE MANUAL

7.1 INTRODUCTION

The following document describes the web services that the third-party software may use to communicate with Ojmar API.

API is divided in 3 different modules, as can be seen in figure below:

1. **Datamodel API:** Permits the interpretation of the information obtained from the locks and cards independently of the version of lock that operates in the facility. This part of the API works as a parser, so it is able to encrypt/decrypt the information of the card/lock.
2. **Recorders' API:** The API implements the communication with both peripherals, including functions to write and read cards and all the NFC programmer functions.
3. **Socket API:** The API implements a mechanism based on sockets to receive information from the locks and to send answers back to them.



Figure 7-1. Figure 1 API Overview.

Document will be divided in these 3 modules.

API will be composed by JSON structures, and 3rd party SW will use the web services in order to communicate with the API.

In most of these web services it is necessary to specify the type of the lock (lock_type) against which the card or the NFC programmer will act. There are different types of locks and they are classified in two groups, locks that can work in the online system and locks that can't work in the online system. Nowadays, there are two Ojmar locks, one for each type working system.

The following table shows different type of locks for each type of working system:

System	value
Offline	OTS
Online	online

In this document, we suppose that the Ojmar API is serving in the following url:
<http://localhost:8086>

7.2 API REFERENCES INDEX

Function	Description	Type	Available
Initialization frame	Creates a frame in order to initialize a Nexo NLX1 lock via NFC programmer	Datamodel	Nexo NLX1
Setup	Configures a lock	Datamodel	Both
Timeset	Creates a frame that sets the time on a lock	Datamodel	Both
Time exchanges	Creates a frame that sets the DST on a lock	Datamodel	OTS family
Automatic opening	Creates a frame that sets an automatic opening on a lock	Datamodel	OTS family
Simple user card	Creates a frame that sets the access rightsaccess rights of a user	Datamodel	Both
Custom user card	Creates a frame that sets custom access rights of a user	Datamodel	Both
Cancellation card	Creates a frame that sets a card UID in the blacklist of the lock	Datamodel	Both
Master card	Creates a frame for a master key	Datamodel	Both
Service card	Creates a frame for a service key	Datamodel	Both
Test card	Creates a frame for a test card	Datamodel	OTS family
Reset card	Creates a frame for a reset card	Datamodel	Both
Events card	Creates a frame to download the events from the lock	Datamodel	OTS family
Subgroup master card	Creates a frame for a subgroup master card	Datamodel	OTS family
Decode frame	Decodes a read frame from a card	Datamodel	Both
Decode events	Decode the events received from a card	Datamodel	OTS family
Define port	Defines the communication port for the NFC programmer on the PC	Recorder	Both

Function	Description	Type	Available
Read card	Reads a card	Recorder	Both
Write card	Writes a created frame on a card	Recorder	Both
Write lock set up data	Uploads the configuration of locks in order to configure them	Recorder	OTS family
Configure lock	Uploads the configuration of locks in order to configure them	Recorder	Nexo NLX1 family
Get events	Downloads the events stored on the NFC programmer	Recorder	OTS family
Update RT OTS FW	Updates the RT OTS FW	Recorder	OTS family
Update OTS Adv. FW	Updates the OTS Adv FW	Recorder	OTS family
Update Nexo NLX1 FW	Updates the Nexo NLX1 FW	Recorder	Nexo NLX1
Update NFC programmer FW	Updates the NFC programmer Fw	Recorder	Nexo NLX1
Asynchronous port configuration	Defines the TCP/IP port communication	Socket	Nexo NLX1
Lock request	Defines the request that a lock makes	Socket	Nexo NLX1
Lock configuration	Gives the option to configure the lock via socket	Socket	Nexo NLX1
Lock operation	Gives the orders that the lock must perform	Socket	Nexo NLX1
Card configuration	Sends card's new configuration via PC	Socket	Nexo NLX1
Lock re configuration	Sends lock's new configuration via PC	Socket	Nexo NLX1
Lock information request	Request information from the lock via PC	Socket	Nexo NLX1

7.3 API REFERENCES DEFINITIONS

7.3.1 Datamodel's API

7.3.1.1 Initialisation & set up

7.3.1.1.1 INITIALIZATION FRAME

Definition

JSON POST file that includes all the necessary information in order to initialize the network of the lock. Only available for the Nexo NLX1 lock.

Parameters

POST <http://localhost:8086/initialization/initData>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String(4) hex	Provided by Ojmar
num_subinstallation	Yes	String(2) hex	
ssid	Yes	String	
password	Yes	String	
encrypt_type	Yes	Integer	From 0 to 8 *
server_ip	Yes	String	
initial_ip	Yes	String	
gateway_ip	Yes	String	
range	Yes	String	
mask	Yes	String	
first_num_lock	Yes	Integer	From 0 to 65534
last_num_lock	Yes	Integer	From 0 to 65534
group	Yes	Integer	From 0 to 255
target	Yes	String	"pp"

*encrypt type values are: 0=OPEN, 1=WEP, 2=WPA_WPA2

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Example

JSON request:

```
{
  7_API REFERENCE MANUAL
```

```

    "num_installation": "B1B2",
    "num_subinstallation": "C2",
    "ssid": "OnlineSystem",
    "password": "A1A2A3A4",
    "encrypt_type": 2,
    "server_ip": "192.168.1.97",
    "initial_ip": "192.168.1.100",
    "gateway_ip": "192.168.1.10",
    "range": "1-50",
    "mask": "255.255.255.0",
    "first_num_lock": 1,
    "last_num_lock": 50,
    "group": 1,
    "target": "pp"
}

```

JSON response:

```

{
    "result": "ok",
    "data":
    "00051111B1B2C2098D020A0401000001000032B1B2030A04010000010000
    321111104090401000001000032C2050B0401000001000032FFFFFFFF145B040
    10000010000324F6E6C696E6553797374656D000000000000000000000000
    000000000000000041314132413341340000000000000000000000000000
    0000000000000000000002C0A80161C0A80164C0A8010A0132FFFFFFFF000A03
    010800FE02411D

```

7.3.1.1.2 SETUP FRAME

Definition

JSON file is sent by POST with the necessary information for the configuration of the lock. Can be used for OTS family & Nexo NLX1 family

Parameters

POST <http://localhost:8086/serialization/setupCard>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS", "online"
num_installation	Yes	String(4) hex	Provided by Ojmar
num_subinstallation	For Nexo NLX1	String(2) hex	
first_num_lock	Yes	Integer	From 0 to 65534
last_num_lock	Yes	Integer	From 0 to 65534
Mode	For OTS family	String	"free" or "dedicated"
card_model_setup	For OTS family	String	"ultralight", "classic" or "desfire"
Subgroup	For OTS family	Integer	From 0 to 15
Group	For Nexo NLX1	Integer	From 0 to 255

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Example

OTS family

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666",
  "first_num_lock": "50",
  "last_num_lock": "73",
  "mode": "free",
  "card_model_setup": "classic",
  "subgroup": 2
}
```

JSON response:

```
{
  "result": "ok",
  "data":
  "666600000D8402000000000000000000000000000000000049000032F0000000000000
  00000000000000000000000000000000000000000000000000000"
}
```

Nexo NLX1 family

JSON request:

```
{
  "lock_type": "online",
  "num_installation": "6666",
  "num_subinstallation": 1,
  "first_num_lock": "50",
  "last_num_lock": "73",
  "group": 1
}
```

JSON response:

```
{
  "result": "ok",
  "data":
  "0005111166660109A7011004010000320000490600020401020102011004
  0100003200004905000204010201060110040100003200004903000200010
  20104011004010000320000490700020101020106080F0401000032000049
  0100000000007F080F040100003200004900000040FC02E00919040100003
  20000490169B89D9CB8D180C969B89D9CB8D180C9090F0401000032000049
  04A0A1A2A3A4A5090F04010000320000490500008627C10A0A03010800FE0
  292C3"
}
```

7.3.1.1.3 FRAME FOR TIME SET

Definition

JSON file is sent by POST with the necessary information time setting of the lock. Can be used for OTS family & Nexo NLX1 family.

- NOTE: The computer where the OjmarAPI is serving must be in time to generate the frames with the correct time.

Parameters

POST <http://localhost:8086/serialization/timeSetCard>

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS", "online"
num_installation	Yes	String(4) hex	
num_subinstallation	For Nexo NLX1	String(2) hex	
Group	For Nexo NLX1	Integer	From 0 to 255
start_range	For Nexo NLX1	Integer	From 0 to 65534
end_range	For Nexo NLX1	Integer	From 0 to 65534

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

OTS family

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666"
}
```

JSON response:

```
{
  "result": "ok",
  "data":
    "66660000300000000000000000000000000000000000093619160815031E0000
    0000000000000000000000000000000000000000000000"
}
```

Nexo NLX1 family

JSON request:

```
{
  "lock_type": "online",
  "num_installation": 6666,
  "num_subinstallation": 1,
  "group": 1,
  "start_range": 15,
  "end_range": 26
}
```

JSON response:

```
{
  "result": "ok",
  "data":
  "00051111666601090E070C040100000F00001A5805F0B70A03010800FE02A4C0"
}
```

7.3.1.1.4 FRAME FOR TIME EXCHANGES

Definition

JSON file is sent by POST with the necessary information for the next 5 time advances and 5 time delays. Can be used for the OTS family.

Parameters

POST <http://localhost:8086/serialization/timeExchangesCard>

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS"
num_installation	Yes	String(4) hex	
time_advances	No (maximum 5 dates)	Array<String>	["dd/mm/yyyy"]
time_delays	No (maximum 5 dates)	Array<String>	["dd/mm/yyyy"]

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Example

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666",
  "time_advances": [
    "28/03/2010",
    "27/03/2011"
  ],
  "time_delays": [
    "31/10/2010",
```

```

        "30/10/2011"
    ]
}

```

JSON response:

```

{
    "result": "ok",
    "data":
    "666600004000E18AD98BFFFFFFFFFFFFFFD0AF50BFFFFFFF8FFFFFFFFF0000
    000000000000000000000000000000000000000000000000"
}

```

7.3.1.1.5 FRAME FOR AUTOMATIC OPENING CARD

Definition

JSON file is sent by POST with the necessary information for the configuration of the automatic opening that will happen every day. Only available for the OTS family.

Parameters

POST <http://localhost:8086/serialization/automaticOpeningCard>

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS"
num_installation	Yes	String(4) hex	
Time	Yes	String	"HH:mm"

Response OK:

```

{
    "result": "ok",
    "data": "XXXXX...XXXX"
}

```

Response NOK:

```

{
    "result": "error",
    "code": "-1",
    "message": "..."
}

```

Example

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666",
  "time": "10:00"
}
```

JSON response:

[illegible]

7.3.1.2 User functionalities

7.3.1.2.1 FRAME FOR SIMPLE USER CARD

Definition

JSON file is sent by POST with the necessary information for the creation of a simple frame for user cards. Available for OTS family & Nexo NLX1.

Parameters

POST http://localhost:8086/serialization/userSimpleCard

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"online","OTS"
num_installation	Yes	String(4) hex	
num_subinstallation	For NEXO NLX 1	String(2) hex	
dedicated_locks	No (maximum 1 lock for OTS family)	Array<Integer>	
free_locks	No (Maximum 3 locks for OTS family)	Integer	
subgroups	For OTS family	Array<Integer>	From 0 to 15

Examples

Example for OTS family

JSON request:

```
{
  "lock type": "OTS",
```

```

    "num_installation": "2222",
    "subgroups": [0, 1, 3],
    "free_locks": 2
}

```

JSON response:

```

{
  "result": "ok",
  "data":
    "2222FFF4060000000000000000FFFFF0000000080FFFF7280FFF800000
    0000000000000000000000000000000000000000000000000000"
}

```

Example for Nexo NLX1

JSON request:

```

{
  "lock_type": "online",
  "num_installation": "2222",
  "num_subinstallation": "89",
  "free_locks": 8,
  "dedicated_locks": [1]
}

```

JSON response:

```

{
  "result": "ok",
  "data":
    "0005111122228901248400000104FFFFFF04FFFFFF04FFFFFF04FFFFFF04
    FFFFFFF04FFFFFF04FFFFFF04FFFFFF03065100006100000A03010800FE02A
    490"
}

```

7.3.1.2.2 FRAMES FOR CUSTOM USER CARD

Definition

JSON file is sent by POST with the information necessary for creating a customized user card. In the case of user cards parameters for OTS family and Nexo NLX1 family are not always the same. Below are two tables with the parameters required for each system.

Parameters

POST <http://localhost:8086/serialization/userSimpleCard>

Parameters for OTS family

Parameter of the request JSON file for the **OTS family** system:

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS"
num_installation	Yes	String(4) hex	
dedicated_locks	No (maximum 1 lock)	Array<Integer>	
free_locks	No (maximum 3 locks)	Integer	From 0 to 3
subgroups	Yes for free locks	Array<Integer>	From 0 to 15
expiry_type	No	String	"no" or "endDate"
expiry_time	If expiry_type = "endDate"	String	"HH:mm"
expiry_date	If expiry_type = "endDate"	String	"dd/mm/yyyy"
prohibition	No (used only with expiry date)	Integer	0 = no (default) 1 = closing forbidden 2 = opening forbidden 3 = both forbidden
time_zones	No (used only with free locks)	Array	
start_time	Yes	String	"HH:mm"
end_time	Yes	String	"dd/mm/yyyy"
weekdays	Yes	Array<Integer>	From 0 to 6 (Monday = 0, Sunday = 6)

Parameters for Nexo NLX1

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"online"
num_installation	Yes	String(4) hex	
num_subinstallatio n	Yes	String(2) hex	
dedicated_locks	No (maximum 1 lock)	Array<Integer>	
free_locks	No	Integer	
expiry_type	No	String	"no", "endDate" or "uses"
expiry_time	If expiry_type = "endDate"	String	"HH:mm"
expiry_date	If expiry_type = "endDate"	String	"dd/mm/yyyy"
start_expiry_time	If expiry_type = "endDate"	String	"HH:mm"
start_expiry_date	If expiry_type = "endDate"	String	"dd/mm/yyyy"
uses	If expiry_type = "uses"	Integer (number of hours)	
profiles	No	Array	
openClose	Yes	Object	
occupy	Yes	Boolean	
close	Yes	String	"USER_CLOSING""AUTO MATIC_CLOSING"
behaviour	Yes	String	"REGULAR BEHAVIOUR""INVERSE BEHAVIOUR"
timeBetweenOpera tions	Yes	Integer	
behaviour	Yes	Object	
habitual	Yes	String	"OCCASIONAL", "MEMBE R"
white_list	Yes	Boolean	
black_list	Yes	Boolean	
timeZones	No	Object	

Parameter	Mandatory	Data type	Possible values
start_date	Yes	String	"dd/mm/yyyy"
start_time	Yes	String	"HH:mm"
end_date	Yes	String	"dd/mm/yyyy"
end_time	Yes	String	"HH:mm"
weekdays	Yes	Array<Integer>	From 0 to 6 (Monday = 0, Sunday = 6)
holidays	Yes	Boolean	
calendar	If holidays = true	Array<Boolean>	(size 368)
group	Yes	Integer	From 0 to 255
start_range	Yes	Integer	From 0 to 65534
end_range	Yes	Integer	From 0 to 65534
time_zone	No (only 1 admitted)	Array	
start_date	Yes	String	"dd/mm/yyyy"
start_time	Yes	String	"HH:mm"
end_date	Yes	String	"dd/mm/yyyy"
end_time	Yes	String	"HH:mm"
weekdays	Yes	Array<Integer>	From 0 to 6 (Monday = 0, Sunday = 6)

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

7.3.1.2.3 EXAMPLES

Example for OTS family

JSON request for offline system with free locks:

```
{
  "num_installation": "6666",
  "lock_type": "OTS",
  "free_locks": 3,
  "subgroups": [0,1,9],
  "expiry_time": "17:00",
  "expiry_date": "19/04/2014",
  "prohibition": 3,
  "time_zones": [{
    "start_time": "09:30",
    "end_time": "17:30",
    "weekdays": [0,1,2,3,4]
  }]
}
```

JSON response:

```
{
  "result": "ok",
  "data":
  "6666FD36000000000000000009A07440000000080FFFF6E80FFFF8080FF
  FF802646070000000000000000000000000000000067"
}
```

JSON request for offline system with dedicated locks:

```
{
  "num_installation": "6666",
  "lock_type": "OTS",
  "dedicated_locks": [294]
}
```

JSON response:

```
{
  "result": "ok",
  "data":
  "66660000050000000000000000000000FFFFF0000000008001265D000000000000
  000000000000000000000000000000000000000000000000"
}
```

JSON request for offline system with locks of two types:

```
{
  "num_installation": "6666",
  "lock_type": "OTS",
  "dedicated_locks": [56],
  "free_locks": 2,
  "subgroups": [6,7],
  "expiry_time": "17:00",
  "expiry_date": "19/07/2014",
  "prohibition": 3,
  "time_zones": [{
    "start_time": "09:30",
    "end_time": "17:30",
    "weekdays": [0,1,2,3,4]
  }]
}
```

JSON response:

```
{
  "result": "ok",
  "data":
  "6666FF3F3300000000000038009B87440000000080FFFF1380FFFF800000
  0000264607000000000000000000000000000067"
}
```

Example for Nexo NLX1 family

JSON request for online system:

```
{
  "lock_type": "online",
  "num_installation": "6666",
  "num_subinstallation": "1",
  "dedicated_locks": [6, 11],
  "free_locks": 3,
  "expiry_type": "endDate",
  "expiry_time": "17:08",
  "expiry_date": "19/04/2014",
  "start_expiry_time": "17:08",
  "start_expiry_date": "19/04/2015",
}
```

```

"profiles": [{
  "openClose": {
    "occupy": true,
    "close": "USER_CLOSING",
    "behavior": "REGULAR_BEHAVIOUR",
    "timeBetweenOperations": 200
  },
  "behavior": {
    "habitual": "OCCASIONAL",
    "white_list": true,
    "black_list": false
  },
  "timeZones": [{
    "start_time": "09:30",
    "start_date": "01/01/2015",
    "end_time": "17:30",
    "end_date": "01/01/2016",
    "weekdays": [0,1,2,3,4]
  }],
  "group": 1,
  "start_range": 1,
  "end_range": 10
}],
"time_zones": [{
  "start_time": "09:30",
  "start_date": "01/01/2015",
  "end_time": "17:30",
  "end_date": "01/01/2016",
  "weekdays": [0,1,2,3,4]
}]
}

```

JSON response:

```

{
  "result": "ok",
  "data":
"0005A1A26666010114840000068400000B04FFFFFFF04FFFFFFF04FFFFFFF02
085533E0F05352AD700303510001040954A513985686B7985C09260112040
100000100000A05000240020201200C14040100000100000A0154A5139856
86B7985C0A030108000B021A52"
}

```

7.3.1.3 Maintenance functionalities

7.3.1.3.1 FRAME FOR CANCELLATION CARD

Definition

JSON file is sent by POST with the information necessary for creating a cancellation card that will send a card to the black list. Available for both OTS family & Nexo NLX1 family.

Parameters

POST <http://localhost:8086/serialization/cancellationCard>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS", "online"
num_installation	Yes	String(4) hex	
num_subinstallation	For Nexo NLX1	String(2) hex	
uid	Yes	String(8) or String(14)	
group	For Nexo NLX1	Integer	From 0 to 255
start_range	For Nexo NLX1	Integer	From 0 to 16777215
end_range	For Nexo NLX1	Integer	From 0 to 16777215

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Examples

Examples for OTS family

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666",
  "uid": "11223344"
}
```

JSON response:

```
{  
    "result": "ok",  
    "data":  
        "666600000A0000000000000000000000000000112233440000004E000000000000  
000000000000000000000000000000000000"  
}
```

Examples for Nexo NLX1

JSON request:

```
{
  "lock_type": "online",
  "num_installation": "1234",
  "num_subinstallation": "33",
  "group": "1",
  "start_range": "1",
  "end_range": "2",
  "uid": "01020304"
}
```

JSON response:

```
{
  "result": "ok",
  "data":
"0005111112343309120E10040100000100000200000001020304000A0301
0800FE025F69"
}
```

7.3.1.3.2 FRAME FOR MASTER CARD

Definition

JSON file is sent by POST with the information necessary for creating a master card that is able to open every lock from an installation. Available for OTS family and Nexo NLX1 family.

Parameters

POST <http://localhost:8086/serialization/masterCard>

Parameter	Mandatory	Data type	Possible values
password	Yes	String	Level1 and Level2 password
lock_type	Yes	String	"OTS", "online"
num_installation	Yes	String(4) hex	
num_subinstallation	For Nexo NLX1	String(2) hex	

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

Example for OTS family

JSON request:

```
{
  "password": "*****",
  "lock_type": "OTS",
  "num_installation": "6666"
}
```

JSON response:

```
{
```

Example for Nexo NLX1 family

```
{
  "lock_type": "online",
  "num_installation": "6666",
  "num_subinstallation": 1,
  "password": "*****"
}
```

```
{
  "result": "ok",
  "data": "0005111166660103037100000A03010800FE026029"
}
```

Response NOK:

Examples

Example for OTS family

JSON request:

JSON response:

Example for Nexo NLX1 family

JSON request:

JSON response:

```
{
  "result": "ok",
  "data": " 0005111166660103033100000A03010800FE021233"
```

}

7.3.1.3.4 FRAME FOR RESET CARD

Definition

JSON file is sent by POST with the information necessary for creating a reset card that restores the lock to factory mode. Available for OTS family and Nexo NLX1 family.

Parameters

POST <http://localhost:8086/serialization/resetCard>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS", "online"
num_installation	Yes	String(4) hex	
num_subinstallation	For the Nexo NLX1	String(2) hex	
bt	No (For the Nexo NLX1)	Boolean	true, false
group	For the Nexo NLX1	Integer	From 0 to 255
start_range	For the Nexo NLX1	Integer	From 0 to 65534
end_range	For the Nexo NLX1	Integer	From 0 to 65534

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Example

Example for OTS family

JSON request:

```
{
  "lock_type": "OTS",
```


JSON response:

```
{
  "result": "ok",
  "data":
"00051111666633090C130A040100000100006401000A03010800FE0247BD"
}
```

7.3.1.3.5 FRAME FOR TEST CARD

Definition

JSON file is sent by POST with the information necessary for creating a test card that open and closes any factory mode lock. Only available for OTS family.

Parameters

POST <http://localhost:8086/serialization/testCard>

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS"

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "... "
}
```

Example

JSON request:

```
{
  "lock_type": "OTS"
}
```

JSON response:

```
{
    "result": "ok",
    "data":
"0000000000F00000000000000000000000000000000000000000000F000000000000
0000000000000000000000000000000000"
}
```

7.3.1.3.6 FRAME FOR EVENTS CARD

Definition

JSON file is sent by POST with the information necessary for creating an events card that can download 3 events from the lock. Available for OTS family.

Parameters

POST <http://localhost:8086/serialization/eventsCard>

JSON file is sent by POST with the information necessary for creating an events card.

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS", "online"
num_installation	Yes	String(4) hex	

Response OK:

```
{
    "result": "ok",
    "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
    "result": "error",
    "code": "-1",
    "message": "..."
}
```

Example

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666"
}
```

JSON response:

```
{  
    "result": "ok",  
    "data":  
        "666600000708000F000000000000000000000000000000000000000000000000  
        00000000000000000000000000000000"  
}
```

7.3.1.3.7 FRAME FOR SUBGROUP MASTER CARD

Definition

JSON file is sent by POST with the information necessary for creating a subgroup master card that open and close any lock from a subgroup. Only available for OTS family.

Parameters

POST http://localhost:8086/serialization/subgroupMasterCard

JSON file is sent by POST with the information necessary for creating a subgroup master card.

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
lock_type	Yes	String	"OTS"
num_installation	Yes	String(4) hex	
subgroups	Yes (maximum 16 subgroups)	Array	From 0 to 15
expiry_type	No	String	"no", "endDate"
expiry_time	If expiry_type = "endDate"	String	"HH:mm"
expiry_date	If expiry_type = "endDate"	String	"dd/mm/yyyy"

Parameter	Mandatory	Data type	Possible values
prohibition	No (used only with expiry date)	Integer	0 = no (default) 1 = closing forbidden 2 = opening forbidden 3 = both forbidden
time_zones	No (used only with free locks)	Array	
start_time	Yes	String	"HH:mm"
end_time	Yes	String	"dd/mm/yyyy"
weekdays	Yes	Array<Integer>	From 0 to 6 (Monday = 0, Sunday = 6)

Response OK:

```
{
  "result": "ok",
  "data": "XXXXX...XXXX"
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

JSON request:

```
{
  "lock_type": "OTS",
  "num_installation": "6666",
  "subgroups": [6, 8],
  "expiry_time": "17:00",
  "expiry_date": "19/07/2014",
  "prohibition": 3,
  "time_zones": [{
    "start_time": "09:30",
    "end_time": "17:30",
    "weekdays": [0,1,2,3,4]
  }]
```

```
    ]]  
}
```

JSON response:

```
{  
  "result": "ok",  
  "data":  
  "6666FEBF3B00000000000000009B874400000000080FFFA20000000000000  
0000264607000000000000000000000000000067"  
}
```

7.3.1.4 Decode frames

7.3.1.4.1 DECODE CARD FRAMES

Definition

JSON file is sent by POST with the frame to decode and return a JSON file with the object information described in the card. Available for OTS family and Nexo NLX1 family

Parameters

POST <http://localhost:8086/serialization/describeCard>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
Frame	Yes	String	

Response OK:

```
{  
  "result": "ok",  
  "card_type": "XXXX",  
  (card information by type ...  
}
```

Response NOK:

```
{  
  "result": "error",  
  "code": "-1",  
  "message": "...."  
}
```

Example

Example for OTS family

JSON request:

```
{
  "frame":
  "808000000D890700000000000000000000000000000049000032F800000000000000
  0000000000000000000000000000000000000000000000000"
}
```

JSON response:

```
{
  "result": "ok",
  "card_type": "setup",
  "num_installation": "8080",
  "first_num_lock": 50,
  "last_num_lock": 73,
  "mode": "free",
  "subgroup": 7,
  "direct_sector": 9
}
```

Example for Nexo NLX1 family

JSON request:

```
{
  "frame": "00051111666633098D020A04010000010000326666030A040100
000100003211110409040100000100003233050B0401000001000032FFFF
F145B04010000010000324F6E6C696E6553797374656D0000000000000000
0000000000000000000000000413141324133413400000000000000000000
00000000000000000000000000000002C0A80161C0A80164C0A8010A0132FFFF
FF000A03010800FE028B69"
}
```

JSON response:

```
{
  "result": "ok",
  "card_type": "init",
  "num_installation": "6666",
  "num_subinstallation": "33",
  "first_num_lock": 1,
  "last_num_lock": 50,
  "ssid": "OnlineSystem",
  "password": "A1A2A3A4",
  "encrypt type": "WPA WPA2",
}
```

```

    "server_ip": "192.168.1.97",
    "initial_ip": "192.168.1.100",
    "gateway_ip": "192.168.1.10",
    "range": "1-50",
    "mask": "255.255.255.0",
    "group": 1,
    "target": "pp"
}

```

7.3.1.4.2 DECODE EVENTS FROM CARD

Definition

JSON file is sent by POST with the frame to decode and return a JSON file with events described in the card. Only available for OTS family.

Parameters

POST <http://localhost:8086/serialization/getEvents>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
frame	Yes	String	

Response OK:

```

{
    "result": "ok",
    "num_installation": XXXX,
    "events": [{
        "num_lock": XX,
        "event_code": XX,
        "UID": "XXXXXXXXXXXX",
        "date": "YYYY/MM/DD HH:mm:ss"
    }]
}

```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

JSON request:

```
{
  "frame":
  "66660003070800401004791AA1431C807A8732881204791AA1431C807A87
  32881004791AA1
  431C807A87328800000000"
}
```

JSON response:

```
{
  "result": "ok",

  "card_type": "events_card",    "num_installation": "6666",
  "num_lock": 3,
  "events": [
    {
      "num_lock": 3, "event_code": "10", "UID": "04791AA1431C80",
      "date": "2014/05/15 12:34:00"
    },
    {
      "num_lock": 3, "event_code": "12", "UID": "04791AA1431C80",
      "date": "2014/05/15 12:34:00"
    },
    {
      "num_lock": 3, "event_code": "10", "UID": "04791AA1431C80",
      "date": "2014/05/15 12:34:00"
    }
  ]
}
```

7.3.2 Recorder's API

7.3.2.1 Define the port for NFC programmer communication

Definition

JSON file is sent by multipart form is sent by POST with the data to set nfc programmer port. Available for the NFC programmer for OTS family & Nexo NLX1 family.

Parameters

MULTIPART FORM POST <http://localhost:8086/nfc/setPort>

Request POST parameters:

Parameter	Mandatory	Data type	Possible values
port	Yes	Text	COMX

Response OK:

```
{
  "result": "ok",
  "data": "Port XX configured correctly",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

JSON request:

```
{
Content-Disposition: form-data; name="port"
COM3
}
```

JSON response:

```
{
  "result": "ok",
  "data": "Port COM3 configured correctly"
}
```

7.3.2.2 Read/write frames from cards with card reader

7.3.2.2.1 READ CARD

Definition

GET request is sent indicating reader type (NFC, CNReader) and the web service will return the frame of the RFID card. Available for OTS & Nexo NLX1 family.

- NOTE: Use describe Card web service in order to translate the information of the received frame.

Parameters

GET http://localhost:8086/card/read/{reader_type}/{card_model}

Request GET parameters:

Parameter	Mandatory	Data type	Possible values
reader_type	Yes	String	"nfc", "cnreader"
card_model	Yes (For OTS family)	String	"ultralight", "classic", "desfire"
card_model	Yes (For Nexo NLX1 family)	String	"classic"

Response OK:

```
{
  "result": "ok",
  "data_ots": "XXXXXX...XXXX" (Opt.)
  "data_online": "XXXXXX.XXXXX", (Opt.)
  "uid": "YYYYYYYY" (4 or 7 bytes)
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Example

JSON request:

<http://localhost:8086/card/read/cnreader/classic>

JSON response:

```
{
  "result": "ok",
  "data_online":
"0005111166663301048400000C03035100000A03010800FE0215E1000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
00",
  "uid": "26F99832"
}
```

7.3.2.2.2 WRITE CARD

Definition

JSON file is sent by POST with the frame and reader type (NFC, CNReader) and the web service will write the frame in the card. Available for OTS & Nexo NLX1 family.

- NOTE: Use Datamodel’s API web services in order to create the frame that will be written on the card.

Parameters

POST <http://localhost:8086/card/write>

Parameters of the request JSON file:

Parameter	Mandatory	Data type	Possible values
reader_type	Yes	String	"nfc", "cnreader"
card_model	Yes (For OTS family)	String	"ultralight", "classic", "desfire"
card_model	Yes (For Nexo NLX1 family)	String	"classic"
frame	Yes	String	

Response OK:

```
{
    "result": "ok",
    "data_ots": "Card write ok" (Opt.)
    "data_online": "Card write ok" (Opt.)
    "uid": "XXXXXXXX" (4 or 7 bytes)
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

JSON request:

```
{
  "reader_type": "cnreader",
  "card_model": "classic",

  "frame": "00051111666633098D020A04010000010000326666030A040100
000100003211110409040100000100003233050B040100000100003200000
1145B0401000001000032496E616B6954657374000000000000000000000000
00000000000000000000000000363636366F6A6D617200000000000000000000
0000000000000000000000000000000002C0A86461C0A86465C0A8640A0132FFFF
FF000A03010800FE0276EB"
}
```

JSON response:

```
{
  "result": "ok",
  "data_online": "Card write ok",
  "uid": "26F99832"
}
```

7.3.2.3 NFC programmer's direct communication with lock

7.3.2.3.1 LOCK CONFIGURATION

Write lock set up data

Definition

JSON file is sent by POST with the information needed to set up locks. Configures locks, its time and the blacklist. Only available for OTS family.

Parameters

POST <http://localhost:8086/nfc/writeInitialization>

Parameter of the request JSON file:

Parameter	Mandatory	Data type	Possible values
Locks	Yes	Array	
Number	Yes	String	
facilityNumber	Yes	String(4) hex	
Subgroup	Yes	String	
Config	Yes	String	"dedicated", "free"
Blacklist	Yes	Array	
UID	Yes	String	Up to 1200 UIDs (4 or 7 bytes)

Response OK:

```
{
  "result": "ok",
  "data": "Init data write ok",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

JSON request:

```
{
  "locks": [
    { "number": 53,      "facilityNumber": "6666",      "subgroup": 1,
      "config": "free" },
    { "number": 1,      "facilityNumber": "6666",      "subgroup": 0,
      "config": "dedicated" }
  ],
  "blacklist": ["12345678"]
}
```

JSON response:

```
{
  "result": "ok",
  "data": "Init data write ok",
}
```

Configure programmer NDM

Definition

Multipart form is sent by POST with the data to update the configuration data of the locks to the programmer. Available for the Nexco NLX1 family

- NOTE: Use the Initialization frame web service or the setupCard web service result as the frame for the upload.

Parameters

MULTIPART FORM POST <http://localhost:8086/nfc/configureProgrammerNMD>

Request POST parameters:

Parameter	Mandatory	Data type	Possible values
data	Yes	String	NDM frame

Response OK:

```
{
  "result": "ok",
  "data": "Ok",
}
```

Response NOK:

```
{
```

```

    "result": "error",
    "code": "-1",
    "message": "...."
}

```

Example

JSON request:

Content-Disposition: form-data; name="data"

```

0005111122223309A70110040100000000006401000204010201020110040
1000000000064020002040102010601100401000000000064030002000102
0104011004010000000000640400020101020102080F04010000010000490
100000000001F080F040100000100004900000040FC02E009190401000001
0000490169B89D9CB8D180C969B89D9CB8D180C9090F04010000010000490
4A0A1A2A3A4A5090F04010000010000490500008627C10A0A030108000B02
8021

```

JSON response:

```

{
    "result": "ok",
    "data": "Ok",
}

```

7.3.2.3.2 LOCK MAINTENANCE

Get events from NFC

Definition

Reads the events that have previously been downloaded from the lock to the NFC programmer. Only available for the OTS family.

Parameters

GET <http://localhost:8086/nfc/getEvents>

The events that have been downloaded to the NFC programmer will be read.

Response OK:

```

[
    {
        "lock_num": XXX,
        "eventCode": XXX,
        "UID": "XXXXXXXXXX",
        "date": "YYYY/MM/DD HH:mm"
    }
]

```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...."
}
```

Example

JSON request:

GET <http://localhost:8086/nfc/getEvents>

JSON response:

```
{
  "result": "ok", "num_installation": "6666", "num_lock": 3,
  "events": [
    {
      "num_lock": 3, "event_code": "10", "UID": "04791AA1431C80",
      "date": "2014/05/15 12:34:00"
    },
    {
      "num_lock": 3, "event_code": "12", "UID": "04791AA1431C80",
      "date": "2014/05/15 12:34:00"
    },
    {
      "num_lock": 3, "event_code": "10", "UID": "04791AA1431C80",
      "date": "2014/05/15 12:34:00"
    }
  ]
}
```

Firmware update

Update the FW of the OTS real time

Definition

Multipart form is sent by POST with the data that updates the Firmware of the real time OTS (From OTS family) lock.

- NOTE: Firmware version must be asked to Ojmar S.A.

Parameters

MULTIPART FORM POST <http://localhost:8086/nfc/reprogramOtsLockFw>

Request POST parameters:

7_API REFERENCE MANUAL

Parameter	Mandatory	Data type	Possible values
file	Yes	File (.bin, .exe)	
version	Yes	String	"X.X"

Response OK:

```
{
  "result": "ok",
  "data": "Ok",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "...",
}
```

Update the FW of the OTS Advance

Definition

Multipart form is sent by POST with the data that updates the Firmware of the OTS Advance (From OTS family) lock

- NOTE: Firmware version must be asked to Ojmar S.A.

Parameters

MULTIPART FORM POST

<http://localhost:8086/nfc/reprogramMultiStandardOtsLockFw>

Multipart form is sent by POST with the data to update the FW.

Request POST parameters:

Parameter	Mandatory	Data type	Possible values
file	Yes	File (.bin)	
version	Yes	String	"X.X"

Response OK:

```
{
  "result": "ok",
  "data": "Ok",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Update the FW of the Nexo NLX1 lock

Definition

Multipart form is sent by POST with the data that updates the Firmware of the Nexo NLX1 lock.

- NOTE: Firmware version must be asked to Ojmar S.A.

Parameters

MULTIPART FORM POST <http://localhost:8086/nfc/reprogramOTSONlineLockFw>

Parameter	Mandatory	Data type	Possible values
file	Yes	File (.bin, .exe)	
version	Yes	String	"X.X.X"

Response OK:

```
{
  "result": "ok",
  "data": "Ok",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Update the FW of the NFC programmer

Definition

Multipart form is sent by POST with the data that updates the Firmware of the NFC programmer. Available for OTS family and Nexo NLX1 family.

- NOTE: Firmware version must be asked to Ojmar S.A.

Parameters

MULTIPART FORM POST <http://localhost:8086/nfc/reprogramNfcProgrammerFw>

Multipart form is sent by POST with the data to update the FW.

Request POST parameters:

Parameter	Mandatory	Data type	Possible values
file	Yes	File(.bin, .exe)	
version	Yes	String	"X.X.X"

Response OK:

```
{
  "result": "ok",
  "data": "Ok",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

7.3.3 Socket API

Nexo NLX1 locks have the ability to work online in a wireless network and establish direct communication with the API. This way the 3rd party SW can receive state information in real time from the locks and determine whether a card should be able to operate with a lock in each situation.

This scenario presents an asynchronous communication where the connection can be started by the lock in any moment.

To make possible this decision delegation the 3rd party SW needs to publish a service for the API to query when a request is received. For that purpose, the API expects a TCP socket to be listening in the machine in a configurable port. The 3rd party SW is responsible to open this socket and implement the protocol to communicate with the online system of the API.

For each connection, the API will forward the information received from the lock translated to JSON format and the 3rd party SW will respond with another JSON.

7.3.3.1 Port configuration

Definition

GET request is sent indicating the port number that third-party software is using for asynchronous communication with the locks. API, by defect, connect to port number 7777.

Parameters

GET http://localhost:8086/online/setPort/{port_num}

Request GET parameters:

Parameter	Mandatory	Data type	Possible values
port_num	Yes	Integer	

Response OK:

```
{
  "result": "ok",
  "data": "Port XX configured correctly",
}
```

Response NOK:

```
{
  "result": "error",
  "code": "-1",
  "message": "..."
}
```

Example

URL:

<http://localhost:8086/online/setPort/56>

JSON response:

```
{  
  "result": "ok",  
  "card_type": "Port 56 configured correctly"  
}
```

7.3.3.2 Lock request

Definition

Every time a card is approached to the lock, this one will send a request to the 3rd party SW with the information of the card and the lock, leaving the decision of what action should be performed to the 3rd party SW.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandatory	Data type	Possible values
profiles	Yes	Array (Int)	
uid	Yes	String	Approached card's UID
mem_uid	Yes	String	UID that lock has in its memory
lock_status	Yes	Array (String)	A - Opened C - Closed L - Free O - Occupied B - Blocked
Num_installation	Yes	String	Lock installation number
Num_subinstallation	Yes	String	Lock subinstallation number
locks	Yes	Array(Int)	Which locks has the card access rights to
lock_request	Yes	Boolean	If lock is making a request
lock_response	Yes	Boolean	If lock is answering a petition
lock_number	Yes	Int	Lock's number

Example

JSON structure with:

```
"profiles":[],
"uid":"000000D4817BAA",
"mem_uid":"000000000000000",
"lock_status":["A","I"],
"num_installation":"2222",
"num_subinstallation":"33",
"locks":[],
"lock_request":true,
"lock_response":false,
"num_lock":1
```

7.3.3.3 3rd party SW answer

Once a request is received, 3rd party SW will be able to answer with the following options (These options can be sent at the same time).

7.3.3.3.1 LOCK CONFIGURATION

Definition

When a Nexa NLX1 lock is initialized, it is connected to the network provided and will send a request to the API. The API will receive the request and forward it to the socket where the 3rd party SW will be listening. If the SW detects that it is the first time receiving information from that lock it will respond with a JSON with the config flag enabled. The API will get the JSON and send back the configuration data to the lock.

First of all, we need to provide the network information to the lock. We will generate an initialization card for that: <http://localhost:8086/initialization/initData>.

Once the initialization card has been used, the lock will get connected to the wireless network and send a request packet to the API at the IP passed in the initialization.

Parameters

JSON structure that the 3rd party SW will answer in order to configure a lock

Parameter	Mandatory	Data type	Possible values
num_lock	Yes	Int	Lock's number
config	Yes	Boolean	Defines if the lock is going to be configured or not.
num_installation	Yes	String	Lock installation number
num_subinstallation	Yes	String	Lock subinstallation number
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	Yes	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000D4817BAA",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "I"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 1
}
```

3rd party SW answer JSON:

```
{
  "num_lock": 1,
  "config": true,
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [2],
  "free_locks": 3,
  "profiles": []
}
```

The lock has been configured.

7.3.3.3.2 LOCK OPENING/CLOSING ALLOWEMENT

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.

Examples

Example that will open/close the lock

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000D4817BAA",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 1
}
```

3rd party SW answer JSON:

```
{
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [1],
  "free_locks": 1
}
```

Example that will not open/close the lock

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000D4817BAA",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 1
}
```

3rd party SW answer JSON:

```
{
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [3]
}
```

3rd party SW will send a `dedicated_lock` that is not the actual lock number. If a different installation or subinstallation number is sent, lock will not perform the action too.

- NOTE: It is mandatory to answer with a no permission JSON in order to avoid the opening or closing of the lock. If there is no answer, lock will try to solve the operation in a local way (As if it was online).

7.3.3.3.3 LOCK OPENING/CLOSING CUSTOM ALLOWEMENT

Definition

3rd party SW will be able to answer if the lock is able to open or close giving to the user custom access rights.

Parameters

Parameter	Mandatory	Data type	Possible values
<code>num_installation</code>	Yes	String	Lock installation number
<code>num_subinstallation</code>	Yes	String	Lock subinstallation number
<code>dedicated_locks</code>	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
<code>free_locks</code>	No	Int	Number of occasional locks that the user will have access rights to.
<code>occupy</code>	Yes	Boolean	Defines if the card occupies the lock when it gets closed
<code>automatic_close</code>	Yes	Boolean	Defines if lock gets auto closed when its opened
<code>reverse_working</code>	Yes	Boolean	Defines if the lock gets occupied when its opened

Parameter	Mandatory	Data type	Possible values
Time_between_operation	Yes	Int	(ms)
Working_type	Yes	String	“dedicated”, “free”
White_list	Yes	Boolean	Defines if the user will be checked in the white list (If offline)
Black_list	Yes	Boolean	Defines if the user will be checked in the black list (If offline)

Examples

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000D4817BAA",
  "mem_uid": "000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 1
}
```

3rd party SW answer JSON:

```
{
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [3]
"occupy": true
automatic_close", false
reverse_working" false
time_between_operations", 100
working_type", "dedicated"
white_list", true
black_list", false
```

}

7.3.3.3.4 LOCK OPENING/CLOSING ALLOWEMENT + CARD CONFIGURATION

Apart from giving the access rights of opening and closing, 3rd party SW can propagate the information to the card so that this one will be configured with its last permission without any human intervention (in case the lock gets offline).

7.3.3.3.5 LOCK OPENING/CLOSING ALLOWEMENT + LOCK CONFIGURATION

Apart from giving the access rights of opening and closing, 3rd party SW can propagate the information to the lock so that this one will be configured with its last permission without any human intervention (in order to work online or offline).

- NOTE: More than one Card configuration + more than one lock configuration can be sent at the same time. (Up to 252 bytes)

Opening/closing + change lock's installation number

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus the option of changing lock's facility number if it is needed.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
change_num_installation	Yes	Array	
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number

Parameter	Mandatory	Data type	Possible values
			that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
num_installation	Yes	String	Lock's new installation number

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000D4817BAA",
  "mem_uid": "000000D4817BAA ",
  "lock_status": ["C", "O"],
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 156
}
```

3rd party SW answer:

```
{
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "dedicated_locks": [1,2],
  "free_locks": 1,
  "change_num_installation": {
    "num_installation": "CCDD",
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

Opening/closing + change lock's subinstallation number

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus the option of changing lock's sub-facility number if it is needed.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
change_num_subinstallation	Yes	Array	
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
num_subinstallation	Yes	String	Lock's new subinstallation number.

```

    "uid": "000000D4817BAA",
    "mem_uid": "000000D4817BAA ",
    "lock_status": ["C", "O"],
    "num_installation": "B1B2",
    "num_subinstallation": "C2",
    "locks": [],
    "lock_request": true,
    "lock_response": false,
    "num_lock": 156
}

```

3rd party SW answer:

```

{
    "num_installation": "B1B2",
    "num_subinstallation": "C2",
    "dedicated_locks": [1,2],
    "free_locks": 1,
    "profiles": [],
    "change_num_subinstallation": {
        "num_subinstallation": "DE",
        "group": 1,
        "range_from": 1,
        "range_to": 100
    }
}

```

Opening/closing + change lock's number

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus the option of changing lock's number if it is needed.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will

Parameter	Mandatory	Data type	Possible values
			have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
change_num_lock	Yes	Array	
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
num_lock	Yes	Int	Lock's new number.

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000D4817BAA",
  "mem_uid": "000000D4817BAA ",
  "lock_status": ["C", "O"],
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 156
}
```

3rd party SW answer:

```
{
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "dedicated_locks": [1,2],
  "free_locks": 1,
  "expiry_type": "no",
  "uses": 0,
  "profiles": [],

```

```

    "change_num_lock": {
      "group": 1,
      "num_lock": 4
    }
  }
}

```

Opening/closing + set lock's time and date

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus setting the lock in time and date.

- NOTE: It is recommended to send this information everytime a daylight saving time happens.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
set_time	Yes	Array	
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "11223344556677",
  "mem_uid": "00000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "AABB",
  "num_subinstallation": "CC",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 1785
}
```

3rd party SW answer JSON:

```
{
  "num_installation": "AABB",
  "num_subinstallation": "CC",
  "dedicated_locks": [5, 8],
  "free_locks": 6,
  "profiles": [],
  "set_time": {
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

Opening/closing + white/black list management

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus managing the white and black list if the lock goes offline.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights

Parameter	Mandatory	Data type	Possible values
			to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
change_lists	Yes	Array	
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
items	Yes	Array	
uid	Yes	String	7 bytes.
action	Yes	String	“add”, “delete”, “modify”.
list_type	Yes	String	“white”, “black”.

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000AABBCCDD",
  "mem_uid": "00000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "EEEE",
  "num_subinstallation": "AC",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 12
}
```

3rd party SW answer JSON:

```
{
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "dedicated_locks": [1,2],
  "free_locks": 1,
  "profiles": [],
  "change_lists": {
    "items": [
      {
        "uid": "000000AAAAAAA", "action": "add",
        "list_type": "white"
      },
      {
        "uid": "000000BBBBBBB", "action": "delete",
        "list_type": "white"
      },
      {
        "uid": "CCCCCCCCDDDDDD", "action": "add",
        "list_type": "black"
      }
    ],
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

Opening/closing + lock reset

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus restoring the lock to factory value.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will

			have access rights to.
profiles	Yes	Array (Int)	
delete_data	Yes	Array	
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
options	Yes	String	"BT".

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000AABBCCDD",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "EEFF",
  "num_subinstallation": "AC",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 12
}
```

3rd party SW answer JSON:

```
{
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "dedicated_locks": [1,2],
  "free_locks": 1,
  "expiry_type": "no",
  "uses": 0,
  "profiles": [],
  "delete_data": {
```

```
    "options": ["BT"],
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

7.3.3.3.6 LOCK OPENING/CLOSING ALLOWEMENT + LOCK CONFIGURATION INFORMATION REQUEST

3rd party SW is able to request information of the lock's configuration in order to receive and manage real time information.

- NOTE: Just one type of information request can be asked per operation.
- NOTE2: Information request can be sent in the same operation alongside with the card configuration and lock configuration.

Opening/closing + lock installation & subinstallation request

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus requesting the information concerning the installation and subinstallation of the lock.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
request_data	Yes	Array	
media	Yes	String	"serial_port".
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
get_inst_subinst	Yes	Boolean	

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000AABBCCDD",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "EEFF",
  "num_subinstallation": "AC",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 12
}
```

3rd party SW answer JSON:

```
{
  "num_lock": 1,
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [2],
  "free_locks": 3,
  "profiles": [],
  "request_data": {
    "media": "serial_port",
    "get_inst_subinst": true,
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

Opening/closing + lock group and number

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus requesting the information concerning the lock's number and group.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	
request_data	Yes	Array	
media	Yes	String	“serial_port”.
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
get_num_lock_group	Yes	Boolean	

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000AABBCCDD",
  "mem_uid": "000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 8
}
```

3rd party SW answer JSON:

```
{
  "num_lock": 8,
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [2],
  "free_locks": 3,
  "profiles": [],
  "request_data": {
    "media": "serial_port",
    "get_num_lock_group": true,
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

Opening/closing + lock installation & lock time and date request

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus requesting the information of the lock's time and date.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
Profiles	Yes	Array (Int)	
request_data	Yes	Array	
media	Yes	String	"serial_port".
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
get_time	Yes	Boolean	

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "123456781234",
  "mem_uid": "123456781234",
  "lock_status": ["C", "O"],
  "num_installation": "6666",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
```

```

    "lock_response":false,
    "num_lock":11
}

```

3rd party SW answer JSON:

```

{
  "num_installation": "6666",
  "num_subinstallation": "33",
  "dedicated_locks": [2],
  "profiles": [],
  "request_data": {
    "media": "serial_port",
    "get_time": true,
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}

```

Opening/closing + get events

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON receiving the last events that happened in the lock

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
profiles	Yes	Array (Int)	

Parameter	Mandatory	Data type	Possible values
request_data	Yes	Array	
media	Yes	String	"serial_port".
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
get_log	Yes	Array	
n_events	Yes	Int	1-10.

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "65849301223344",
  "mem_uid": "65849301223344",
  "lock_status": ["O", "B"],
  "num_installation": "2432",
  "num_subinstallation": "16",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 6157,
}
```

3rd party SW answer JSON:

```
{
  "num_lock": 1,
  "num_installation": "2432",
  "num_subinstallation": "16",
  "dedicated_locks": [2],
  "free_locks": 3,
  "profiles": [],
}
```

```

    "request_data": {
        "media": "serial_port",
        "get_log": {
            "n_events": 8
        },
        "group": 45
        "range_from": 6000,
        "range_to": 6200
    }
}

```

Opening/closing + get number of cycles

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus requesting the cycles that the lock has already performed.

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
Profiles	Yes	Array (Int)	
request_data	Yes	Array	
media	Yes	String	"serial_port".
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number

Parameter	Mandatory	Data type	Possible values
			that will be affected (From 1 to 65534).
get_n_cycle	Yes	Boolean	

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000BCDEFABC",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 3
}
```

3rd party SW answer JSON:

```
{
  "num_lock": 1,
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [2],
  "free_locks": 3,
  "profiles": [],
  "request_data": {
    "media": "serial_port",
    "get_n_cycle": true,
    "group": 1,
    "range_from": 1,
    "range_to": 100
  }
}
```

Opening/closing + get public configuration

Definition

3rd party SW will be able to answer if the lock is able to open or close with that card by answering the request with the following JSON plus requesting the complete configuration of the lock

Parameters

Parameter	Mandatory	Data type	Possible values
num_installation	Yes	String	Lock installation number.
num_subinstallation	Yes	String	Lock subinstallation number.
dedicated_locks	Yes	Array(Int)	Dedicated locks that the user will have access rights to.
free_locks	No	Int	Number of occasional locks that the user will have access rights to.
Profiles	Yes	Array (Int)	
request_data	Yes	Array	
media	Yes	String	"serial_port".

Parameter	Mandatory	Data type	Possible values
group	Yes	Int	Group of locks that will be affected (From 1 to 254).
range_from	Yes	Int	Initial lock number that will be affected (From 1 to 65534).
range_to	Yes	Int	Last lock number that will be affected (From 1 to 65534).
read_public_information	Yes	Boolean	

Example

Lock request JSON:

```
{
  "profiles": [],
  "uid": "000000BCDEFABC",
  "mem_uid": "0000000000000000",
  "lock_status": ["A", "L"],
  "num_installation": "2222",
  "num_subinstallation": "33",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 3
}
```

3rd party SW answer JSON:

```
{
  "num_lock": 1,
  "num_installation": "2222",
  "num_subinstallation": "33",
  "dedicated_locks": [2],
  "free_locks": 3,
  "profiles": [],
  "request_data": {
    "media": "eth",
```

```

        "read_public_information": true,
        "group": 1,
        "range_from": 1,
        "range_to": 100
    }
}

```

7.3.3.4 Lock response

Every 3rd party SW answer will be replied by the lock sending a final response. This response can contain two different structures. First one will be for all the opening/closing operation + card configuration + lock configuration and the last one will be the answer to the information request.

7.3.3.4.1 LOCK RESPONSE TO OPENING/CLOSING PETITION (INCLUDING CARD CONFIGURATION AND LOCK CONFIGURATION)

Definition

Lock will always answer the last event that happened in the lock, that includes event type, uid that performed the action, time and date that happened.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandatory	Data type	Possible values
Log_items	Yes	Array ()	
log_item	Yes	Array(string)	
Timestamp	Yes	String	Unix format.
eventCode	Yes	String	Event code + description (See Annex).
battery_status	Yes	String	% of battery level.

Example

```

{
  logItems=[LogItem [timestamp=1476885376, uid=00000667BE4404
, eventCode=EventType[USER_CLOSE: 18], batteryStatus=73]]
}

```

7.3.3.4.2 LOCK RESPONSE TO OPENING/CLOSING PETITION (INCLUDING CARD CONFIGURATION AND LOCK CONFIGURATION) + INFORMATION REQUEST

Lock installation & subinstallation request

Returns the information concerning the installation and subinstallation data of the lock, plus the last event.

Definition

Lock will always answer the last event that happened in the lock, that includes event type, uid that performed the action, time and date that happened and the lock's installation and subinstallation information.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandatory	Data type	Possible values
Log_items	Yes	Array ()	
log_item	Yes	Array(string)	
Timestamp	Yes	String	Unix format.
eventCode	Yes	String	Event code + description (See Annex).
battery_status	Yes	String	% of battery level.
InformationRequest	Yes	Array	
Information	Yes	String	LockNumGroupResponse.
num_installation	Yes	String	Lock's facility number.
Num_subinstallation	Yes	String	Lock's sub installation number.

Example

```
logItems=[LogItem [timestamp=1476886236, uid=00000667BE4404
, eventCode=EventType[USER_OPEN: 16],
batteryStatus=73]]], commands=[CommandCont
ainer [commands=[InformationRequest
[information=NumInstSubinstResponse[sub inst
alation=51, instalation=26214]]]]]
```

Lock number and group request

Returns the information concerning the lock number and group, plus the last event.

Definition

Lock will always answer the last event that happened in the lock, that includes event type, uid that performed the action, time and date that happened and the lock's installation and subinstallation information.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandatory	Data type	Possible values
Log_items	Yes	Array ()	
log_item	Yes	Array(string)	
Timestamp	Yes	String	Unix format.
eventCode	Yes	String	Event code + description (See Annex).
battery_status	Yes	String	% of battery level.
InformationRequest	Yes	Array	
Information	Yes	String	LockNumGroupResponse.
group	Yes	String	Lock's group.
ext	Yes	String	Lock's extension.
Number	Yes	String	Lock's number.

Example

```
logItems=[LogItem [timestamp=1476887561, uid=00000667BE4404
, eventCode=EventType[USER_CLOSE: 18],
batteryStatus=73]]], commands=[CommandCon
tainer [commands=[InformationRequest
[information=LockNumGroupResponse[group=1,
ext=0, number=1]]]]]
```

Lock events request

Definition

Returns the information concerning the last events of the lock, plus the last event.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandatory	Data type	Possible values
Log_items	Yes	Array ()	
log_item	Yes	Array(string)	
Timestamp	Yes	String	Unix format.
eventCode	Yes	String	Event code + description (See Annex).
battery_status	Yes	String	% of battery level.
InformationRequest	Yes	Array	E
Information	Yes	String	LogResponse.
*Log_items	Yes	Array ()	
*log_item	Yes	Array(string)	
*Timestamp	Yes	String	Unix format.
*eventCode	Yes	String	Event code + description (See Annex).
*battery_status	Yes	String	% of battery level.

*Per number of events

Example

```
logItems=[LogItem [timestamp=1476888500, uid=0000000000000000
, eventCode=EventType[ALARM_CAM_FORCED: 96],
batteryStatus=100]]],commands=[Com
mandContainer [commands=[InformationRequest
[information=LogResponse[overwrite=0
, logSize=69, logItems=[LogItem [timestamp=1476888500,
uid=0000000000000000, event
Code=EventType[ALARM_CAM_FORCED: 96], batteryStatus=100],
LogItem [timestamp=147
6888501, uid=00000667BE4404, eventCode=EventType[USER_CLOSE:
18], batteryStatus=
73], LogItem [timestamp=1476888500, uid=00000667BE4404,
eventCode=EventType[OPER
ATION_REQUEST: 7], batteryStatus=73], LogItem
[timestamp=1476887991, uid=0000066
7BE4404, eventCode=EventType[USER_OPEN: 16],
batteryStatus=73], LogItem [timesta
```

```

mp=1476887990, uid=00000667BE4404,
eventCode=EventType[OPERATION_REQUEST: 7], ba
tteryStatus=73], LogItem [timestamp=1476887561,
uid=00000667BE4404, eventCode=Ev
entType[USER_CLOSE: 18], batteryStatus=73], LogItem
[timestamp=1476887560, uid=0
0000667BE4404, eventCode=EventType[OPERATION_REQUEST: 7],
batteryStatus=73], Log
Item [timestamp=1476886236, uid=00000667BE4404,
eventCode=EventType[USER_OPEN: 1
6],
batteryStatus=73]]]]]]],dataModelVersion=DataModelVersion[vIn
teger=0, vDecim
al=3,
vSecDecimal=18],dataModelCheck=DataModelChk[checksum=50397],]
{"log":{"overwrite":false,"size":69,"events":[{"date":1476888
500,"event_code":"6
0","uid":"0000000000000000","battery_status":100},{"date":14768
88501,"event_code":
"12","uid":"00000667BE4404","battery_status":73},{"date":1476
888500,"event_code"
:"7","uid":"00000667BE4404","battery_status":73},{"date":1476
887991,"event_code"
:"10","uid":"00000667BE4404","battery_status":73},{"date":147
6887990,"event_code
":"7","uid":"00000667BE4404","battery_status":73},{"date":147
6887561,"event_code
":"12","uid":"00000667BE4404","battery_status":73},{"date":14
76887560,"event_cod
e":"7","uid":"00000667BE4404","battery_status":73},{"date":14
76886236,"event_cod
e":"10","uid":"00000667BE4404","battery_status":73}]}],

```

Lock number of cycles

Definition

Returns the information concerning the cycles that the lock has already completed.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandatory	Data type	Possible values
Log_items	Yes	Array ()	
log_item	Yes	Array(string)	
Timestamp	Yes	String	Unix format.
eventCode	Yes	String	Event code + description (See Annex).
battery_status	Yes	String	% of battery level.
InformationRequest	Yes	Array	
Information	Yes	String	NumberOfCyclesResponse.
fromStartUp	Yes	Int	
fromBatteryChange	Yes	Int	

Example

```
logSize=69, logItems=[LogItem [timestamp=1476888906,  
uid=00000667BE4404  
, eventCode=EventType[USER_OPEN: 16],  
batteryStatus=73]]], commands=[CommandCont  
ainer [commands=[InformationRequest  
[information=NumberOfCyclesResponse[fromStar  
tUp=7602432, fromBatteryChange=7602432]]]]]
```

Lock public information

Definition

Returns the information concerning the complete configuration that the lock has.

Parameters

A JSON structure will be received by TCP/IP with following parameters.

Parameter	Mandator y	Data type	Possible values
Log_items	Yes	Array ()	
log_item	Yes	Array(string)	
Timestamp	Yes	String	Unix format.
eventCode	Yes	String	Event code + description (See Annex).
battery_status	Yes	String	% of battery level.

Parameter	Mandatory	Data type	Possible values
InformationRequest	Yes	Array	
Information	Yes	String	ReadPublicConfigurationResponse
inst	Yes	String	
subinst	Yes	String	
time	Yes	Int	Unix time.
battery_status	Yes	Int	
Hw_status	Yes	String	
fromStartUp	Yes	Int	
fromBatteryChange	Yes	Int	
UID	Yes	String	UID that is occupying the lock.

```
logItems=[LogItem [timestamp=1476889166, uid=00000667BE4404
, eventCode=EventType[USER_CLOSE: 18],
batteryStatus=73]]], commands=[CommandCon
tainer [commands=[InformationRequest
[information=ReadPublicConfigurationRespons
e[integrator=0,inst=0,subinst=0,time=0,id8=0,id7=0,id6=0,id5=
0,id4=0,id3=0,id2=0
,id1=0,dstH=0,dstha=0,dsthd=0,dstDa=0,dstDd=0,lowBatteryPos=0
,batteryStatus=0,hw
Status=0,profiles=0,tmpProfiles=0,fromStartUp=0,fromBatteryCh
ange=0,uid=00000000
0A0300]]]]]
ation":0,"time":"01/01/1970","openings":[],"default_pos_low_b
attery":0,"battery_
status":0,"hw_status":0,"profiles":[],"tmp_profiles":[],"cycl
es_from_init":0,"cy
cles_from_change":0,"uid":"000000000A0300"},
```

7.4 EXAMPLE OF HOW TO CONFIGURE LOCK FROM FACTORY MODE

This example will explain how to start working with a lock once it is received from Ojmar S.A, starting from the initialization & configuration, creating a user card and allowing it to close/open the lock and receiving 2 events as a maintenance operation.

7.4.1 Lock Complete Configuration

7.4.1.1 Creation of a initialization frame

An initialization frame will be created using the following web service. Following data will be introduced:

POST <http://localhost:8086/initialization/initData>

JSON request:

```
{
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "ssid": "OnlineSystem",
  "password": "A1A2A3A4",
  "encrypt_type": 2,
  "server_ip": "192.168.1.97",
  "initial_ip": "192.168.1.100",
  "gateway_ip": "192.168.1.10",
  "range": "1-50",
  "mask": "255.255.255.0",
  "first_num_lock": 1,
  "last_num_lock": 50,
  "group": 1,
  "target": "pp"
}
```

JSON response:

```
{
  "result": "ok",
  "data":
    "00051111B1B2C2098D020A0401000001000032B1B2030A04010000010000
    32111104090401000001000032C2050B0401000001000032FFFFFF145B040
    10000010000324F6E6C696E6553797374656D0000000000000000000000
    0000000000000000413141324133413400000000000000000000000000
    0000000000000000000002C0A80161C0A80164C0A8010A0132FFFFFF000A03
    010800FE02411D"
}
```

Once the frame is created, this data will be loaded to the NFC programmer:

MULTIPART FORM POST <http://localhost:8086/nfc/configureProgrammerNMD>

JSON request:

Content-Disposition: form-data; name="data"

```
00051111B1B2C2098D020A0401000001000032B1B2030A040100000100003
2111104090401000001000032C2050B0401000001000032FFFFFFF145B0401
0000010000324F6E6C696E6553797374656D00000000000000000000000000
0000000000000004131413241334134000000000000000000000000000000
000000000000000002C0A80161C0A80164C0A8010A0132FFFFFFF000A030
10800FE02411D
```

JSON response:

```
{
  "result": "ok",
  "data": "Ok",
}
```

So, initialization data has been uploaded to the programmer.

7.4.1.2 Initialize the lock and configure it by the socket

Choose the lock option on the NFC programmer, Nexo NLX1 and initialize option. Choose the lock number that wants to be initialized, and approach the programmer to the lock:

Once done, following data will be sent from the lock to the server.

```
{
  "uid": "05814719",
  "mem_uid": "00000000",
  "lock_status": ["A", "L"],
  "num_installation": "B1B2",
  "num_subinstallation": "C2",
  "locks": [],
  "lock_request": true,
  "lock_response": false,
  "num_lock": 1
}
```

The third-party software reads the JSON, detects if it is the first time that this lock makes a request and sends the following JSON forcing configuration.

```
{
    "num_lock":1,
    "config":true,
    "change_lists":{
        "group":"1",
        "range_from":"0",
        "range_to":1,
        "items":[
            {
                "uid":"94FC6B4A",
                "action":"add",
                "list_type":"white"
            },
            {
                "uid":"B4512B4A",
                "action":"add",
                "list_type":"white"
            }
        ]
    },
    "set_time":{
        "group":"1",
        "range_from":"0",
        "range_to":1
    },
    "profiles":null,
    "num_installation":"B1B2",
    "num_subinstallation":"C2",
    "dedicated_locks":[0],
    "free_locks":0
}
```

Lock will send back the event in order to finish the communication and will beep 3 times.

```
{
    "logs":[
        {
            "date":1460390756,
            "event_code":"64",
            "uid":"000094FC6B4A04",
            "battery_status":33
        }
    ]
}
```

7.4.2 User card request

7.4.2.1 Record a user card

Once the lock is configured, a user card will be written. For that, two steps will be made

POST <http://localhost:8086/serialization/userSimpleCard>

JSON request:

```
{
  "lock_type": "online",
  "num_installation": "2222",
  "num_subinstallation": "89",
  "free_locks": 8,
  "dedicated_locks": [1]
}
```

JSON response:

```
{
  "result": "ok",
  "data":
"00051111B1B2C201248400000104FFFFFFF04FFFFFFF04FFFFFFF04FFFFFFF04
FFFFFFF04FFFFFFF04FFFFFFF04FFFFFFF03065100006100000A03010800FE02A
490"
}
```

POST <http://localhost:8086/card/write>

JSON request:

<http://localhost:8086/card/read/cnreader/classic>

JSON response:

```
{
  "result": "ok",
  "data_online":
"00051111B1B2C201248400000104FFFFFFF04FFFFFFF04FFFFFFF04FFFFFFF04
FFFFFFF04FFFFFFF04FFFFFFF04FFFFFFF03065100006100000A03010800FE02A
490",
  "uid": "26F99832"
}
```

7.4.2.2 User card closing allowement

A user card is approached to the lock and lock sends following JSON to the socket.

```
{
  "uid": "94FC6B4A",
```

```

    "mem_uid":"00000000",
    "lock_status":["A","L"],
    "num_installation":"2222",
    "num_subinstallation":"33",
    "locks":[
        {
            "lock_type":"dedicated",
            "num_lock":"1"
        }
    ],
    "lock_request":true,
    "lock_response":false,
    "num_lock":1
}

```

The third party software reads the JSON and sends the following JSON, closing the lock and requesting logs.

```

{
    "num_lock":1,
    "config":false,
    "request_data":{
        "group":"1",
        "range_from":"0",
        "range_to":1,
        "media":"serial_port",
        "get_log":{ "n_events":2 }
    },
    "profiles":null,
    "num_installation":"B1B2",
    "num_subinstallation":"C2",
    "dedicated_locks":[1],
    "free_locks":6
}

```

The lock detects that the third-party software ask for logs, so sends the following JSON.

```
{
    "logs":[
        {
            "date":1460390756,
            "event_code":"12",
            "uid":"000094FC6B4A04",
            "battery_status":33
        },
        {
            "date":1460390754,
            "event_code":"10",
            "uid":"000094FC6B4A04",
            "battery_status":33
        }
    ],
    "num_installation":"2222",
    "num_subinstallation":"33",
    "locks":[
        {
            "lock_type":"free",
            "num_lock":"1"
        }
    ],
    "lock_request":false,
    "lock_response":false,
    "num_lock":0
}
```

7.5 CODE EXAMPLES

7.5.1 PHP

```
// Create a TCP Stream socket
$sock = socket_create(AF_INET, SOCK_STREAM, 0);

// Bind the socket to an address/port
socket_bind($sock, '0.0.0.0', 7777) or die('Could not bind to
address');

// Start listening for connections
socket_listen($sock);
```

7_API REFERENCE MANUAL

```
// Accept incoming requests and handle them as child
processes
$client = socket_accept($sock);

// Read the input from the client
$input = socket_read($client, 65535);

// Process request and write response to $output.

// Write response
socket_write($client, $output, strlen($output));

// Close the master sockets
socket_close($sock);
```

7.5.2 Java

```
public class TcpExample {
    private Socket socket;

    public static void main (String[] args) {
        // Port is assumed to come as first argument
        Socket socket = new Socket("127.0.0.1", args[1]);

        // Pre-cook response
        List<String> whitelist = new ArrayList<String>();
        whitelist.add("A1B2C3D4");
        whitelist.add("A1B2C3D4");
        whitelist.add("A1B2C3D4");
        whitelist.add("A1B2C3D4");

        // Receive data
        byte[] data = readBytes();

        // Send response
        Gson gson = new Gson();
        String rawResponse = gson.toJson(whitelist);
        sendBytes(rawResponse.getBytes());
    }
}
```

```

    public void sendBytes(byte[] myByteArray) throws
IOException {
        sendBytes(myByteArray, 0, myByteArray.length);
    }

    public void sendBytes(byte[] myByteArray, int start, int
len) throws IOException {
        if (len < 0)
            throw new IllegalArgumentException("Negative
length not allowed");
        if (start < 0 || start >= myByteArray.length)
            throw new IndexOutOfBoundsException("Out of
bounds: " + start);
        // Other checks if needed.

        // May be better to save the streams in the support
class;
        // just like the socket variable.
        OutputStream out = socket.getOutputStream();
        DataOutputStream dos = new DataOutputStream(out);

        dos.writeInt(len);
        if (len > 0) {
            dos.write(myByteArray, start, len);
        }
    }

    public byte[] readBytes() throws IOException {
        // Again, probably better to store these objects
references in the support class
        InputStream in = socket.getInputStream();
        DataInputStream dis = new DataInputStream(in);

        int len = dis.readInt();
        byte[] data = new byte[len];
        if (len > 0) {
            dis.readFully(data);
        }
        return data;
    }

```

```
    }  
}
```

7.5.3 C

```
using System;  
using System.Net;  
using System.Net.Sockets;  
using System.Text;  
  
public class SynchronousSocketClient {  
  
    public static void StartClient() {  
        // Data buffer for incoming data.  
        byte[] bytes = new byte[2048];  
  
        // Connect to a remote device.  
        try {  
            // Establish the remote endpoint for the socket.  
            // This example uses port 7777 on the local  
computer.  
            IPEndPoint ipHostInfo =  
            Dns.Resolve(Dns.GetHostName())  
                .IPAddressList[0];  
            IPEndPoint remoteEP = new  
            IPEndPoint(ipHostInfo, 7777);  
  
            // Create a TCP/IP socket.  
            Socket sender = new  
            Socket(AddressFamily.InterNetwork,  
                SocketType.Stream, ProtocolType.Tcp );  
  
            // Connect the socket to the remote endpoint.  
            Catch any errors.  
            try {  
                sender.Connect(remoteEP);  
  
                Console.WriteLine("Socket connected to {0}",  
                    sender.RemoteEndPoint.ToString());  
            }  
            catch (SocketException e) {  
                Console.WriteLine("SocketException: {0}", e);  
            }  
        }  
        catch (Exception e) {  
            Console.WriteLine("Exception: {0}", e);  
        }  
    }  
}
```

```

        // Receive the response from the remote
device.

        int bytesRec = sender.Receive(bytes);
        Console.WriteLine("Received: {0}",

Encoding.ASCII.GetString(bytes, 0, bytesRec));

        // Encode the data string into a byte array.
        byte[] whitelist =
Encoding.ASCII.GetBytes("['A1B2C3D4E5', 'A1B2C3D4E5', 'A1B2C3D4
E5']");

        // Send the data through the socket.
        int bytesSent = sender.Send(whitelist);

        // Release the socket.
        sender.Shutdown(SocketShutdown.Both);
        sender.Close();

    } catch (ArgumentNullException ane) {
        Console.WriteLine("ArgumentNullException :
{0}", ane.ToString());
    } catch (SocketException se) {
        Console.WriteLine("SocketException :
{0}", se.ToString());
    } catch (Exception e) {
        Console.WriteLine("Unexpected exception :
{0}", e.ToString());
    }
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}

public static int Main(String[] args) {
    StartClient();
    return 0;
}
}

```

7.6 ANEXO

7.6.1 Generate and read JSON files

7.6.1.1 PHP

PHP has a function to create JSON string from an array. `string json_encode (mixed $value [,int $options = 0 [, int $depth = 512 ]])`

Example:

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' =>
5);
echo json_encode($arr);
?>
```

The above example would be: `{"a":1,"b":2,"c":3,"d":4,"e":5}`

PHP has another function to decode JSON string and return an array. `mixed json_decode ( string $json [, bool $assoc = false [, int $depth = 512 [, int $options = 0]]])`

Example:

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json, true));
?>
```

The above example would be:

```
array(5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}
```

7.6.1.2 Java

In JAVA there are many JSON libraries to work with JSON strings. This document will show the usage of one of these libraries, Gson.

Create JSON string from an object.

```
DataObject obj = new DataObject();
Gson gson = new Gson();
// convert java object to JSON format, and returned as JSON
formatted string
String json = gson.toJson(obj);
```

Read data from JSON file and convert to object.

```
Gson gson = new Gson();
BufferedReader br = new BufferedReader(new
FileReader("c:\\file.json"));
//convert the json string back to object
DataObject obj = gson.fromJson(br, DataObject.class);
```

7.6.1.3 C

In C# it is possible to create a JSON string from an object.

```
var jsonSerializer = new
System.Web.Script.Serialization.JavaScriptSerializer();
string json = jsonSerializer.Serialize(yourCustomObject);
```

It is possible to create an object from a JSON string.

```
var jsonSerializer = new
System.Web.Script.Serialization.JavaScriptSerializer();
var yourCustomObject =
jsonSerializer.DeserializeObject<MyType>(jsonString);
```

7.6.2 How to make a REST call

7.6.2.1 PHP

7.6.2.1.1 GET REQUEST

```
$curl = curl_init();
curl_setopt($curl, CURLOPT_URL, "http://...");
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
$curl_response = curl_exec($curl);
curl_close($curl);
$decoded = json_decode($curl_response);
```

7.6.2.1.2 POST REQUEST

```
$curl = curl_init();
curl_setopt($curl, CURLOPT_URL, "http://...");
$curl_post_data = array(
    'data1' => 'xxxxxxx',
    'data2' => 'xxxxxx'
);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
curl_setopt($curl, CURLOPT_POST, true);
curl_setopt($curl, CURLOPT_POSTFIELDS, $curl_post_data);
$curl_response = curl_exec($curl);
curl_close($curl);
$decoded = json_decode($curl_response);
```

7.6.2.1.3 POST JSON STRING

```
$curl = curl_init();
curl_setopt($curl, CURLOPT_URL, "http://...");
$jsonString = json_encode($data);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "POST");
curl_setopt($curl, CURLOPT_POSTFIELDS, $jsonString);
curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
$curl_response = curl_exec($curl);
curl_close($curl);
$decoded = json_decode($curl_response);
```

7.6.2.2 Java

This document shows how to create a RESTful JAVA client with Java build-in HTTP client library. We will use "java.net.URL" and "java.net.HttpURLConnection".

7.6.2.2.1 GET

```
URL url = new URL("http://...");
HttpURLConnection conn = (HttpURLConnection)
url.openConnection(); conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
if (conn.getResponseCode() != 200) {
    throw new RuntimeException("Failed : HTTP error code : "
+ conn.getResponseCode());
}
```

```

BufferedReader br = new BufferedReader(new
InputStreamReader(( conn.getInputStream() )));
String output;
System.out.println("Output from Server .... \n");
while ((output = br.readLine()) != null) {
    System.out.println(output);
}
conn.disconnect();

```

7.6.2.2.2 POST

```

URL url = new URL("http://....");
URLConnection conn = (URLConnection)
url.openConnection();
conn.setDoOutput(true);
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/json");
String input = "{\"qty\":100,\"name\":\"iPad 4\"}";
OutputStream os = conn.getOutputStream();
os.write(input.getBytes());
os.flush();
if (conn.getResponseCode() != HttpURLConnection.HTTP_CREATED)
{
    throw new RuntimeException("Failed : HTTP error code : "
+ conn.getResponseCode());
}
BufferedReader br = new BufferedReader(new
InputStreamReader((conn.getInputStream())));
String output;
System.out.println("Output from Server .... \n");
while ((output = br.readLine()) != null) {
    System.out.println(output);
}
conn.disconnect();

```

7.6.2.3 C

Using the code is pretty straightforward. You just create an instance of the `RestClient` class, assign the value of your endpoint (the endpoint is the URL of the REST service you are attempting to call), and call the `MakeRequest` method.

Basic call:

```
string endPoint = @"http:\\myRestService.com\\api\\";
var client = new RestClient(endPoint);
var json = client.MakeRequest();
```

If you want to append parameters you can pass them into the make request method like so.

```
var json = client.MakeRequest("?param=0");
```

To set the `HttpVerb` (i.e. GET, POST), simply use the provided `HttpVerb` enumeration. Here is an example of making a POST request:

```
var client = new RestClient(endpoint: endPoint,
                             method: HttpVerb.POST,
                             postData:      "{ 'someValueToPost':
'The Value being Posted' }");
```

You can also just assign the values in line if you want:

```
var client = new RestClient();
client.EndPoint = @"http:\\myRestService.com\\api\\"; ;
client.Method = HttpVerb.POST;
client.PostData = "{postData: value}";
var json = client.MakeRequest();
```

The `HttpWebRequest` object is in the `System.Net` namespace

PAGE LEFT BLANK INTENTIONALLY

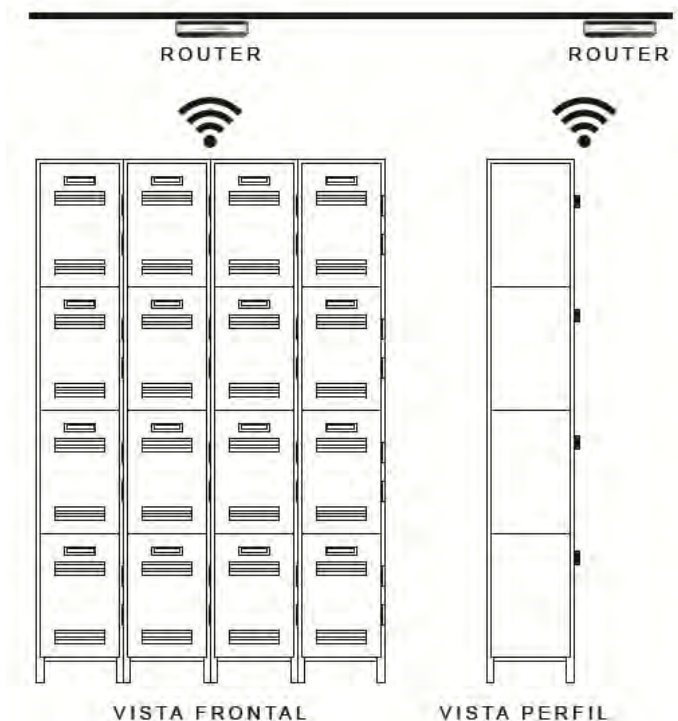
8. ANNEXES

8.1 PHYSICAL INSTALLATION OF THE ROUTER

It is necessary to connect the router to the mains and Ethernet connection is also necessary.

The best position of the router is on the ceiling in front of the locks (this position depends on the layout of the lockers in the changing room). Below are the different lockout layouts with the best position of the router:

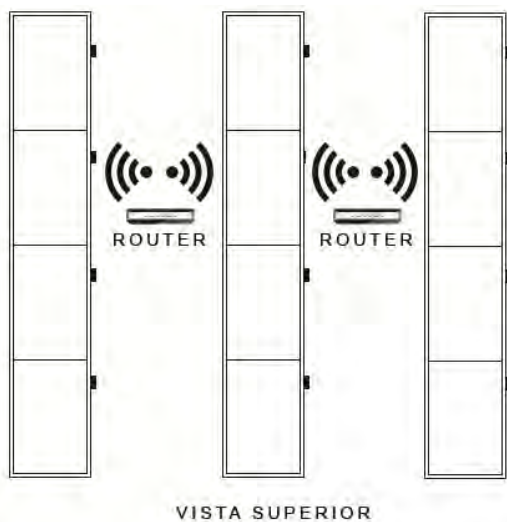
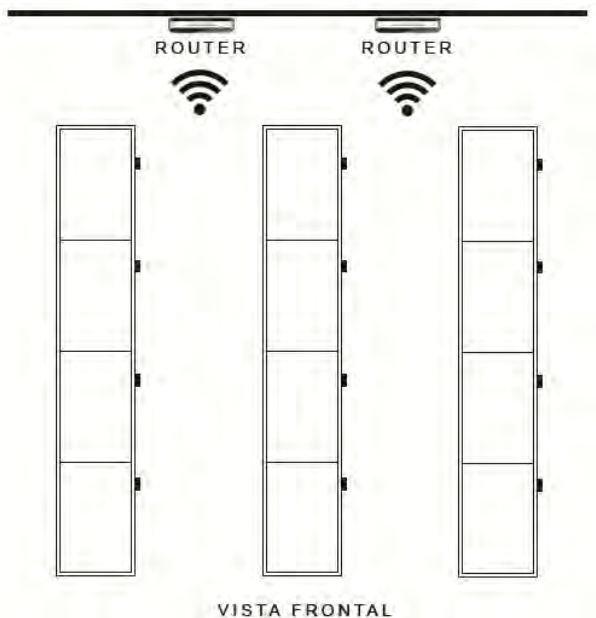
- Option a)



- Option b)



- Option c)



- INSTALLING ONE ROUTER PER 50 LOCKS IS RECOMMENDED.



INTELLIGENT LOCKING SYSTEMS

OJMAR, S.A.
Polígono Industrial de Lerun, s/n
20870 Elgoibar - Gipuzkoa
SPAIN
T. +34 943 748484
www.ojmar.com

