



## **WP3 – Adapt-&-Play Seamless Integration of Legacy Equipment and Building Systems**

Document Version:

### **D3.1 Technical upgrades and integration mechanism for legacy equipment – Initial version**

5.0

<b>Project</b>	<b>Project</b>	
<b>Number:</b>	<b>Acronym:</b>	<b>Project Title:</b>
893079	PHOENIX	Adapt-&- <u>P</u> lay Holistic <u>c</u> ost- <u>E</u> ffective and user-frie <u>N</u> dly <u>I</u> nnovations with high replicability to upgrade smartness of e <u>X</u> isting buildings with legacy equipment

<b>Contractual Delivery Date:</b>	<b>Actual Delivery Date:</b>	<b>Deliverable Type* - Security**:</b>
31/05/2020	01/06/2020	R – PU

\* Type: P – Prototype, R – Report, D – Demonstrator, O – Other

\*\* Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)



<b>Responsible and Editor/Author:</b>  Antonio Skarmeta	<b>Organization:</b>  UMU	<b>Contributing WP:</b>  WP3
---	---------------------------------	------------------------------------

**Authors (organizations):**

UMU, OdinS, ARDEN, UBITECH, SAGOE and LTU

**Abstract:**

This document describes the initial mechanisms for integration of legacy equipment designed and used within the Proof of Concept. The strategies for increasing connectivity include the installation of new middleware, the replacement of systems and the software connection of existing devices to the general platform. This document is a first version of a catalogue of integration mechanisms that will be the result of this work package.

**Keywords:**

Smartness; Integration; Legacy equipment; Communication; Middleware.

**Disclaimer:** The present report reflects only the authors' view. The European Commission is not responsible for any use that may be made of the information it contains.

# Revision History

The following table describes the main changes done in the document since created.

Revision	Date	Description	Author (Organization)
0.1	29/03/2021	Preparation of the skeleton of the document	UMU
0.3	30/04/2021	First version with all content	ODINS / UMU
1.0	05/04/2021	Integration of mechanisms	ODINS
2.0	10/04/2021	Contextualisation of interventions	UMU
3.0	19/05/2021	Incorporation of text by UMU and new restructuration of the document	UMU
4.0	31/05/2021	Comments from first reviewer	ARDEN
5.0	31/05/2021	Comments from Second reviewer	MERITCH

# Executive Summary

This document is the first part of a set of deliverables that are going to describe the mechanisms of integration of legacy equipment within the PHOENIX platform for the duration of the project. This deliverable represents the Initial version of the “*Technical upgrades and integration mechanism for legacy equipment*”. This is a technical document, that explains the integration mechanisms that have been selected to make sure that legacy equipment and the so called quasi-smart devices were able to send the data to the PHOENIX platform. This effort implied tackling the issue from two fronts (1) the installation of new sensing and actuating equipment, and (2) the development of design-for-purpose gateways to enable communication of existing equipment. For this effort we have considered gateway as any solution either physical or software-based that enables communication to and from the platform.

The actuations described in this document have been implemented in the PoC. However, as explained here, a great deal of consideration of the other pilots have been taken to make sure that the lessons learnt in this preliminary work are useful for the larger interventions that are planned in other pilots. This is presented also in this document where a plan of integration has been done for each pilot thanks to the mechanisms learnt on the PoC.

A series of conclusions are presented at the end of this document. Some of the most noticeable are the way in which the integration of systems can be very diverse. Also, we have seen how the quasi-smart devices, namely systems that have connectivity capabilities but are not fully integrated on an intelligent platform, can be sometimes difficult to integrate, and that together with the fact that electronics’ costs are low, it makes more cost effective at times to replace systems that present difficulties for integration.

## Disclaimer

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 893079, but this document only reflects the consortium’s view. The European Commission is not responsible for any use that may be made of the information it contains.



# Table of Contents

<b>1.</b>	<b><i>Introduction.....</i></b>	<b>9</b>
1.1.	Scope of the Document.....	9
1.2.	Relevance to other deliverables.....	9
1.3.	Structure of the document.....	9
<b>2.</b>	<b><i>Description of the PoC as a representative site of the pilots.....</i></b>	<b>10</b>
2.1.	Legacy equipment and systems.....	10
2.2.	Preliminary study on potential intervention of other pilots that can benefit from the PoC .....	11
<b>3.</b>	<b><i>Integrations of legacy equipment and systems in the PoC .....</i></b>	<b>12</b>
3.1.	Solar hot water .....	13
3.2.	Electric vehicles' charging point circuit.....	15
3.3.	HVAC of the building .....	16
3.4.	Air Quality monitoring .....	18
3.5.	WiFi Smart Sockets.....	20
3.6.	PV Plant inverter.....	21
<b>4.</b>	<b><i>Integration of external weather data via APIs for the PoC.....</i></b>	<b>22</b>
<b>5.</b>	<b><i>Building Management System (BMS) in the UMU PoC.....</i></b>	<b>23</b>
<b>6.</b>	<b><i>Manuals for the integration of hardware and software.....</i></b>	<b>23</b>
<b>7.</b>	<b><i>Design of agents to integrate on the broker .....</i></b>	<b>24</b>
7.1.	FIWARE Smart Data Model.....	24
<b>8.</b>	<b><i>Integration plan for the pilots based on PoC learnings.....</i></b>	<b>26</b>
8.1.	UMU .....	26
8.2.	Miwenergía .....	26
8.3.	KaMa .....	27

8.4.	LTU.....	27
8.5.	ARDEN.....	28
9.	<i>Conclusions and further work</i> .....	29
10.	<i>Bibliography</i> .....	31
	<i>Example JSON-LD data model</i> .....	32
	<i>Annex I - Raspberry Pi Gateway for Z-Wave devices</i> .....	33
	<i>Annex II - Modbus Devices (RTU and TCP)</i> .....	41
	<i>Annex III - WiFi Tasmota devices Example of Smart Sockets</i> .....	45
	<i>Annex IV - Interoperable APIs provided by the PHOENIX platform</i> .....	49
	<i>Annex V – Example of WiFi Multi-Sensors for CO<sub>2</sub>, temperature, humidity</i> .....	57



## **1. Introduction**

### **1.1. Scope of the Document**

This document is the Initial version of a set of three deliverables that describe the upgrades and integration of legacy equipment that is carried out as project PHOENIX evolves. This initial version includes the large effort done by the consortium for the development of the Proof of Concept pilot (PoC). This includes the integration of legacy equipment via the creation of novel gateway-solutions and the connection to quasi-smart devices thanks to middle-ware that connects to different communication mechanisms. In addition, the deliverable includes an integration plan for the rest of the pilots. The lessons learnt on the PoC have served to design this integration strategy and they suggest different mechanisms that have been seen to be optimal.

### **1.2. Relevance to other deliverables**

This deliverable is related to the rest of the set namely Deliverable 3.2 and 3.3. The integration mechanisms that are described in this document will have to be developed in the following months of the project. This integration on the main pilots will be described on the intermediate and final version of the integration deliverable.

Although the aim of this document is the description of the integration of devices, one should take into consideration Deliverable 1.4, which defined the “Data management plan” of the project. In the document, the plan on how to manage the data that will come from these sensors is described.

### **1.3. Structure of the document**

The document includes ten sections and a series of appendixes. The introduction gives a contextualisation of this document, the description of the PoC gives a detailed explanation of the site in which the PoC was completed, as well as providing information on why the site is representative of the situations that will be found in the full pilots. This is followed by Section 3 in which the effort that has been made to install the new hardware is described. This includes a description of the external data sources that have been integrated for the PoC and a brief description of them. Section 5 goes into detail about the integration of the existing BMS of the pilot site onto the PHOENIX platform using a designed for purpose middleware.

The description of the manuals that are attached to this main document as annexes is given in Section 6. Section 7 then describes the entities that have been created to enable the integration of legacy equipment, with an emphasis on the developments on data models that has been developed to send the data on NGSI-LD format for each case.

Finally, Section 8 describes the integration plans for the different pilots after the analysing of the learnings that were given by the integration on the PoC, and the document finishes with Conclusions and References.

## **2. Description of the PoC as a representative site of the pilots**

The PoC has been deployed in a sub-set of buildings of the University of Murcia, in the Campus of Espinardo. There are several buildings where a number of devices are already operational. Among them, some are already integrated in an existing BMS while others are essentially isolated and unable to communicate with any platform. One of the goals of this PoC is to add connectivity to these isolated devices.

In addition to having all the devices connected, another goal of this pilot is to have a single platform where all the different equipment can speak the same language to simplify the maintenance tasks of each system.

### **2.1. Legacy equipment and systems**

As part of the PoC, the legacy equipment involved was:

- Solar domestic hot water.
- Electric vehicle chargers.
- HVAC of Pleiades building.
- PV Installation by the Campus' swimming pool.
- Air Quality monitoring.
- WiFi Smart Sockets.
- Building Management System (BMS)

In addition, there are other legacy devices/systems yet to be integrated, and are planned to be considered for the following month in which the UMU Pilot will be deployed:

- Smart electric vehicle charging points. There are some (at least one for now) charging points in the Campus compatible with OCPP (Open Charging Point Protocol).
- In the Pleiades building there is a BMS that's managing a number of devices. It has direct control over individual air conditioning units, temperature, humidity, lighting, IR presence sensors, etc.
- The same building has 2 external backup petrol generators.

## 2.2. Preliminary study on potential intervention of other pilots that can benefit from the PoC

The intervention performed on the PoC was the result of a great deal of evaluation of the potential that each single measurement was going to have on the full project. This included a full study of the services that each pilot has on their definition of the pilot, the field in which it falls within the SRI framework, and the possibility of testing it on the PoC. This information is shown in Table 1.

**Table 1. Table of services planned in each pilot and its feasibility in the PoC with the integrations performed.**

From pilot	Field	Service on GA	PoC feasible?
<b>KaMa</b>	Energy savings	Increase energy efficiency + PV and ST	Yes
<b>KaMa</b>	Grid flexibility	Reduce grid dependency	No
<b>KaMa</b>	Self-generation	Generation of renewable 20kW	Yes
<b>KaMa</b>	Comfort	Dynamic envelope, Automatic shading	No
<b>KaMa</b>	Convenience	Smart energy storage (batteries)	No
<b>KaMa</b>	Information to occupants	interfaces to assess real time performance	Yes
<b>ARDEN</b>	Energy savings	Improved control	Yes
<b>ARDEN</b>	Grid flexibility	Load shifting	Yes
<b>ARDEN</b>	Comfort	Ambient temperatures	Yes
<b>ARDEN</b>	Information to occupants	Monitoring and Control	Yes
<b>UMU</b>	Comfort	Crowdsensing thermostat	Yes
<b>UMU</b>	Grid flexibility	Peak reduction optimisation	Yes
<b>UMU</b>	Grid flexibility	Optimisation of EV charging, cooling, heating and DHW (peak)	Yes
<b>UMU</b>	Grid flexibility	Optimisation of generation consumption and storage based on energy market (energy)	Yes
<b>Miwenergía</b>	Energy saving onsite	Optimisation of conditioning	Yes
<b>Miwenergía</b>	Energy saving on site	Improve efficiency of lighting	Yes
<b>Miwenergía</b>	Flexibility for the grid	Load shifting with gateways and actuators	Yes
<b>Miwenergía</b>	Comfort	Ambient sensors for better internal spaces	Yes
<b>Miwenergía</b>	Convenience	Systems of operational control	Yes
<b>Miwenergía</b>	Information to occupants	Monitoring and control to report info to occ	Yes
<b>LTU</b>	Energy saving on site	Monitoring of energy consumption	Yes
<b>LTU</b>	Comfort	Monitoring of internal conditions	Yes
<b>LTU</b>	Health and wellbeing	Monitoring of ventilation operation	Yes
<b>LTU</b>	Energy Saving on site	Optimisation of conditioning	Yes
<b>LTU</b>	Energy saving on site	Improving energy efficiency of lighting	Yes
<b>LTU</b>	Comfort	Indoor sensors to improve conditions	Yes
<b>LTU</b>	Convenience	Operational control	Yes
<b>LTU</b>	Energy saving on site	Smart thermostats	Yes

LTU	Information to users	Monitoring and control + demand side management	Yes
-----	----------------------	---	-----

In addition to the overall objective of the project of implementing services, it was important to validate that the integrations performed were going to contribute to improving the Smart Readiness Indicator of the PoC. For that, a detailed study of the potential integrations was carried out, comparing the impact that they may have on the overall SRI. Each one of the domains was recalculated with the smartness delivered by each integration, and the extra points were accounted. They were considered as a cumulative rise summing up all domains on the SRI score.

In addition to that, a subjective scale of “PHOENIX interest” was assigned for each integration to account for the extra services that could be gained with that given integration. That scale went from 1 to 5, and contributed to the decision. With the two scales, and taking into consideration the cost of each intervention, it was seen which ones were the most cost effective actuations on the PoC. Table 2 is a summary of these.

**Table 2. Evaluation of all potential interventions that could be done in the PoC.**

Domain SRI code	Hardware	Location	SRI points gained (acc)	PHOENIX interest
<b>Electric Vehicle</b>	Meter and Actuator on UT charging line	UT location	10	5/5
<b>Electric Vehicle</b>	Connection to Circutor chargers	Faculty of CS	10	5/5
<b>DHW</b>	Heat meter for solar DHW installation	Ground floor shed near parking	9	5/5
<b>Electricity</b>	Smart sockets in all appliances	Scattered over the rooms	9	3/5
<b>Controlled ventilation</b>	Air quality sensors (CO2)	Scattered over the building	8	5/5
<b>Controlled ventilation</b>	Actuation over MVHR	PLEIADES Rooftop	7	5/5
<b>Electricity</b>	Petrol generator monitoring	Ground floor shed near parking	7	5/5
<b>Electricity: Renewable &amp; storage</b>	Meters and actuators security shed	Security at University entrance	5	5/5
<b>Heating</b>	VRF power meter	PLEIADES roof top	3	5/5
<b>DHW</b>	Smart sockets on 30 domestic water tanks	Scattered over the building	3	3/5
<b>Cooling</b>	VRF power meter	PLEIADES roof top	3	5/5
<b>Lights</b>	Remote ON/OFF of lights	Central switch at entrance	3	3/5
<b>Monitoring and Control</b>	Occupancy control (IR sensors)	Scattered over the rooms	2	2/5
<b>Heating</b>	Presence control	Each room	1	1/5
<b>Cooling</b>	Presence control	Each room	1	1/5

### 3. Integrations of legacy equipment and systems in the PoC

The site in which the PoC had a range of devices that have a significant relevance for the smartness of the building and for its energy consumption, but that were not monitored or connected by any means. It was important therefore to adapt that equipment to enable their monitoring and control using the PHOENIX platform. The situation of the equipment before the intervention and after the intervention, is summarised in Table 3 with a detailed description in the following subsections.

**Table 3. Situation of Legacy equipment before and during PoC.**

<b>System</b>	<b>Situation prior PoC</b>	<b>Situation on PoC</b>
Solar Domestic Hot Water	No connection	Generation of Power by the Solar panels monitored. Flow of water in the secondary circuit monitored. Temperature of operation IN/OUT monitored
Electric Vehicles Charing Points	No connection	Monitoring of the energy consumed at the charging points. Possible actuation for charging scheduling
Conditioning system + AHU	No connection	Monitoring of consumption on conditioning and AHU. Monitoring of temperatures in relevant points of the system
Indoor environment sensing	Local SCADA communication	Integration into PHOENIX platform
Sockets Loads	No connection	Monitoring of loads and potential actuation
PV plant yield	Logging into local (isolated) server	Secured connection to PHOENIX platform

### **3.1. Solar hot water**

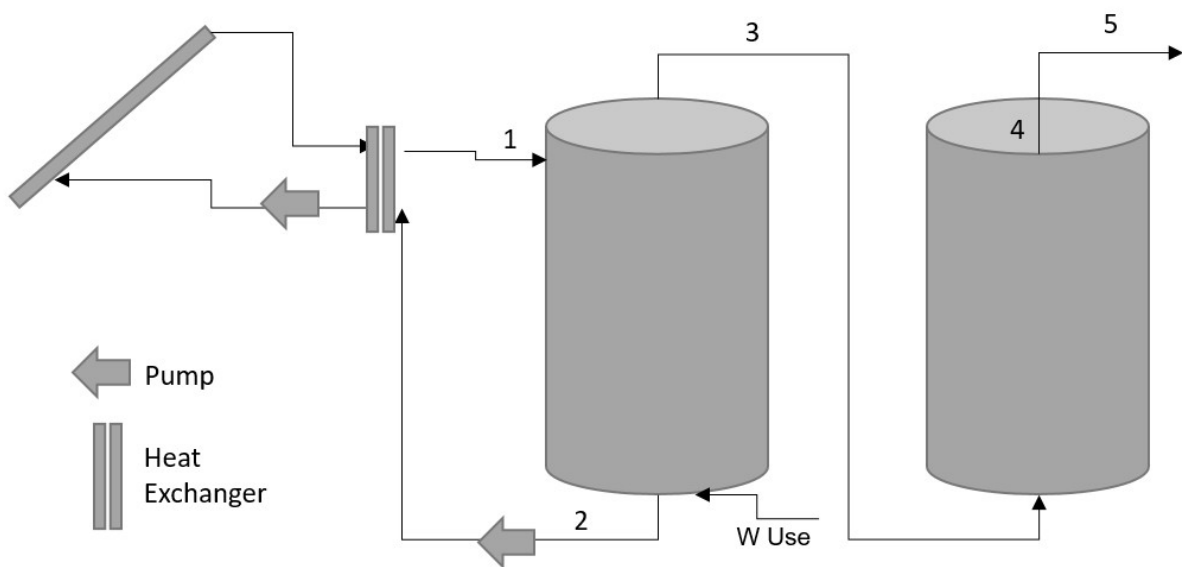
The solar system is shown on Figure 1. The water from the large tank (SOLAR) is heated at the heat exchanger and the other (DHW's sanitary hot water) is heated when there is water demand with hot water coming from SOLAR on one side. The other tank has an auxiliary electric resistance heater that is activated when its temperature drops below a certain level, in case there is no water demand or in case there was so much that the solar panels were sufficient to heat the water in the SOLAR tank.

A possible added value was identified in monitoring the consumption of the circuit from which

the electric resistance heater is connected to know when it is activated.

### Measuring and actuation capacities

- Dual tank system (SOLAR + DHW)
- Multiple temperature reading points
  - SOLAR tank  $\leftrightarrow$  solar panel circuit (1) (2)
  - SOLAR tank  $\rightarrow$  DHW tank (3)
  - DHW tank  $\rightarrow$  building (4)
- Water consumption meter (global counter and real time flow meter) (5)



**Figure 1. Diagram of the domestic hot water system.**





**Figure 2. Photographs of the installations performed.**

#### Hardware and Software connectivity

- Tanks: SOLAR and DHW
- Controller: data reading + integration with PHOENIX MQTT IoT-Agent
- Temperature probes
- Water meter

As can be seen in Figure 2, the tanks are rather large in the actual installation. Moreover, the finished installation required modification at the hydraulic level to install both the temperature probes and the meter.

### **3.2. Electric vehicles' charging point circuit**

The PoC has a point of charge for low-weight electric vehicles. It is worth emphasising that after the intervention of the PoC actuation is available on this charging points, so you can restrict the time ranges in which the charging of vehicles is restricted to those in which electricity is cheaper or greener (or any other criteria).





**Figure 3. Example of electric vehicle used on the PoC.**

#### Measuring and actuation capacities

- Existing electric circuit
- Power metering
- LOCAL/AUTO mode
- Actuation in AUTO mode



**Figure 4. Installation of meters and controllers for the electric cars' charging point.**

#### Hardware and Software connectivity

- Initial installation: one circuit with two charging points
- Second electric control panel with controller, power meter and relays (data reading + actuation + integration with PHOENIX MQTT IoT-Agent)
- LOCAL/AUTO selector

### **3.3. HVAC of the building**

The building has a centralised conditioning system consisting of a variable refrigerant flow (VRF)



system with individual consoles in the internal zones. The system is a TOSHIBA unit with three compressors on the roof, and the consoles in the areas vary from 2 to 4 kW of nominal power. In addition to that, the building has an air handling unit (AHU) with a variety of sensors that control the ventilation in the interior zones. The AHU has a heat recovery unit, to reduce the energy consumption of fresh air supply.



**Figure 5. Photograph of the room of the building with the compressors of the conditioning system on the left and the Air Handling Unit in the right.**

#### Measuring and actuation capacities

Power metering of three different electric circuits, one for AHU and two for conditioning.

- Air conditioning electric panel.
- Ventilation electric panel with LOCAL/AUTO selectors.
- Air handling unit.
- Controller, connection to power meter and relays (data reading + actuation + integration with PHOENIX MQTT IoT-Agent).



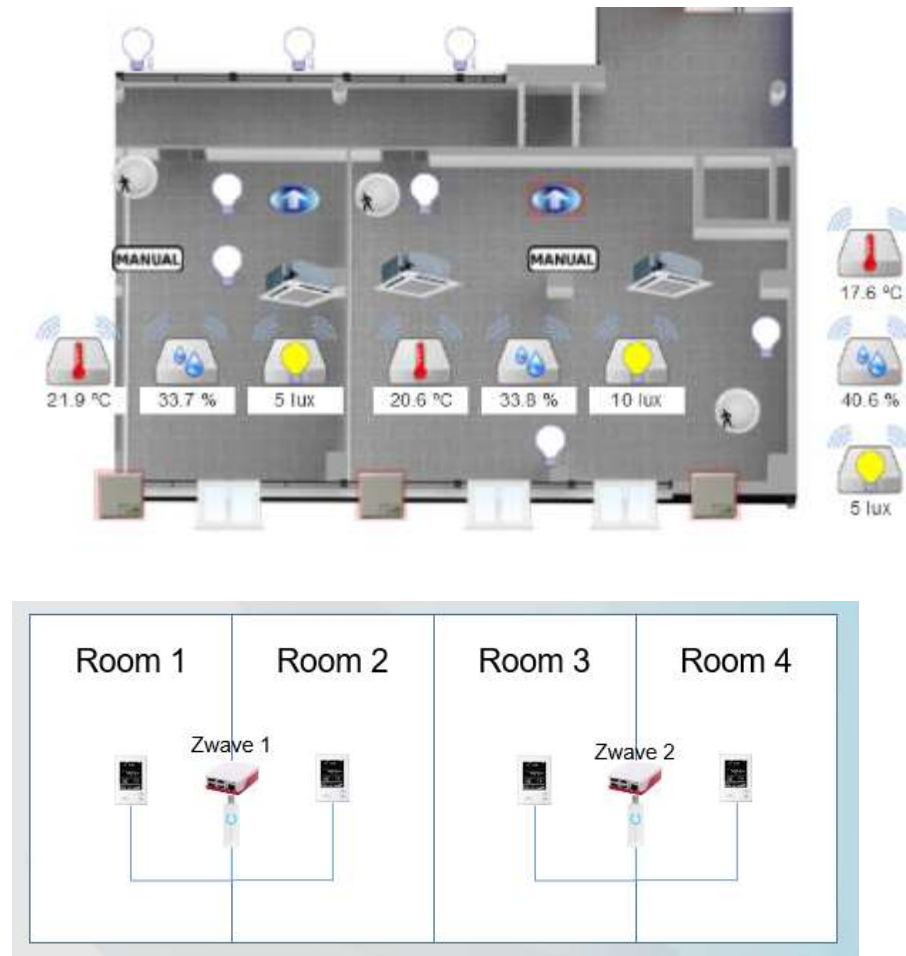
**Figure 6. Installations performed for air conditioning,**

### **3.4. Air Quality monitoring**

Air quality is one of the most important ways of achieving comfort. For this reason, monitoring of Air quality has been crucial in this PoC. In addition to the common parameters measured to account for comfort such as temperature, humidity and lighting, several CO<sub>2</sub> sensors have been installed in the PoC to monitor AQ in a broader sense, and with it, opening the door to the design and testing of new smart services oriented to improve comfort and wellbeing.

The CO<sub>2</sub> sensors communicated on Z-Wave, presenting an ideal opportunity to integrate a popular protocol as in the field of building automation. For the connection of these devices to the PHOENIX platform, a designed-for-purpose gateway was created using a Raspberry-Pi, to conduit their data.

While a single Raspberry Pi might be sufficient for range at which the CO<sub>2</sub> devices are operating, two have been configured and each only listens to the sensors to which it is associated. It should be mentioned that the impact associated with installing the sensors and the gateways is minimal because they are wireless. The information they submit will be used to make decisions within the service that controls the ventilation system within the PHOENIX solution.



**Figure 7. Monitoring systems for air quality.**

#### Hardware and Software connectivity

- Raspberry Pi with Z-Wave adapter in Room 1
- Raspberry Pi with Z-Wave adapter in Room 3
- Z-Wave CO2 sensor
- Integration with PHOENIX IoT Agents supporting extended security capabilities



**Figure 8. CO<sub>2</sub> monitor (left) and RaspberryPi-mounted connectivity system (right).**

### **3.5. WiFi Smart Sockets**

Consumption behaviour is very important for the purpose of the project in terms of informing building users, enabling improved efficiency and increasing building smartness. A way of identifying work patterns of building users was important for this purpose. After a great deal of examination of the potential options, smart sockets were accepted like the best solution. A set of 35 smart sockets were installed on devices that represent occupants' behaviour such as computer monitors, café machine, or printers.

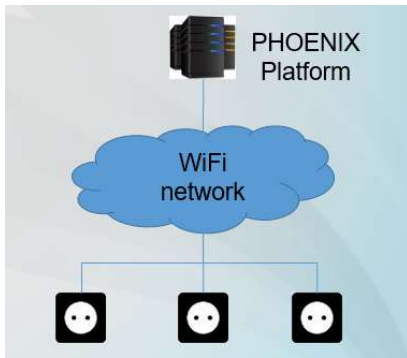
To connect these devices a Tasmota firmware was configured so it can send the data over the WiFi network towards the MQTT agent of the PHOENIX platform, where the conversion to the NGSI-LD data format can take place.

#### Measuring and actuation capacities

- Power metering
- Socket status
- Local/Remote actuation

#### Hardware and Software connectivity

- Custom IoT Agent adapting the MQTT topic format used by the smart socket (Tasmota firmware) to the PHOENIX MQTT IoT Agent format



**Figure 9.** The smart sockets connect to the PHOENIX platform through the WiFi network.

### 3.6. PV Plant inverter

The ten photovoltaic solar installations on the University of Murcia Espinardo campus and the Alamo Fountain Technology Centre have a production capacity of 3,627,128 kWh/year, if all production were used for self-consumption, 16% of the institution's energy needs could be supplied, according to data from a technical report from the Electricity and Electronics Section of the Department of Estates and the 2015 consumption report at the institution. These figures provide an overview of the capacity of renewables in the site of the PoC and the needs of UMU for clean energy sourcing.

PHOENIX will connect to one of these installations, and the procedure to do so was to connect via an FTP connection and to download the quasi-real time data onto the PHOENIX platform.

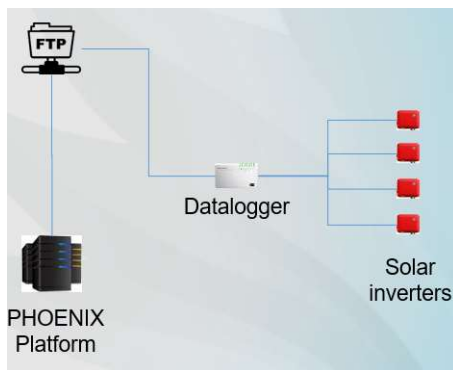


**Figure 10.** Installation on the PoC site.



### Measuring and actuation capacities

The DataLogger (Sunny WebBox) is wired via a bus with inverters and is physically installed in the basement of the pool. Periodically and autonomously it sends the collected data to the FTP server and the "driver" that is running on the platform is responsible for consuming it.



To integrate the PV information, a middleware component was developed to enable the access to the FTP Server and send this information to the PHOENIX hub using a MQTT/SSL interface. The main HW/SW components that participate in this integration are indicated as it follows:

### Hardware and Software connectivity

- Datalogger sends data from solar inverters to FTP server
- Service executed by *cron* in FTP server moves zip files to the destination folder (year/month) based on file name (wb150009187.YYYYMMDD-hhmmss.zip)
- Service executed by *cron* in PHOENIX Platform consumes new data from FTP server and sends it to PHOENIX MQTT IoT Agent.

## **4. Integration of external weather data via APIs for the PoC**

For the PoC, the integration of external data sources of weather was rather important for the purpose of the project in order to leverage real-time data and to forecast weather conditions on the PoC's site. To achieve that, a well proven weather data source called Weatherbit was integrated. The Weatherbit API<sup>1</sup> provides simple weather data access to the Weatherbit.io service. With just latitude and longitude coordinates, you can get weather forecast data returned in JSON format. It has a free plan for 125 API requests per day. Further details are contained in the Weatherbit.io

---

<sup>1</sup> <https://rapidapi.com/weatherbit/api/weather>

API Specification document.

A middleware component is being developed to enable the access to the APIs of the data source and send this information to the PHOENIX hub using a MQTT/SSL interface. This information has been modelled also on NGSI-LD forma with the following IDs:

Weather Observed: urn:ngsi-ld:WeatherObserved:Spain-WeatherObserved-Murcia-LaVereda
WeatherForecast (day 1): urn:ngsi-ld:WeatherForecast:Spain-WeatherForecast-Daily-1-Murcia-LaVereda
...
WeatherForecast (day 16): urn:ngsi-ld:WeatherForecast:Spain-WeatherForecast-Daily-16-Murcia-LaVereda
WeatherForecast (hour 1): urn:ngsi-ld:WeatherForecast:Spain-WeatherForecast-Hourly-1-Murcia-LaVereda
...
WeatherForecast (hour 48): urn:ngsi-ld:WeatherForecast:Spain-WeatherForecast-Hourly-48-Murcia-LaVereda

## 5. Building Management System (BMS) in the UMU PoC

In the buildings use in the PoC at the University of Murcia (UMU), there is a Building Management System (BMS) based on the IoT platform "OpenData" which provides a multi-user SCADA-based web technology to centralize all sensor information. The kind of data collected is related to environmental parameters such as temperature, humidity and lighting and occupancy in the rooms. The sensors and BMS data can be accessed via two enablers of the FIWARE platform that has been created and deployed. The Orion Context Broker (OCB) processes each new sensor measurement and the COMET enabler is a repository for the historic data of each sensor. The following figure presents the BMS architecture with sensors, actuators, gateways deployed in the buildings and the cloud data management server with different user interfaces. Moreover, the BMS provides REST/JSON APIs to exchange real-time sensor/actuation data with the Orion Broker, as well as to retrieve historical data from the COMET repository. To allow the integration and data exchange between the BMS system and the PHOENIX platform, in T3.1, two middleware components have been implemented to translate the data format and from BMS APIs with the NGSI-LD interface of the PHOENIX hub described on Annex IV.

## 6. Manuals for the integration of hardware and software

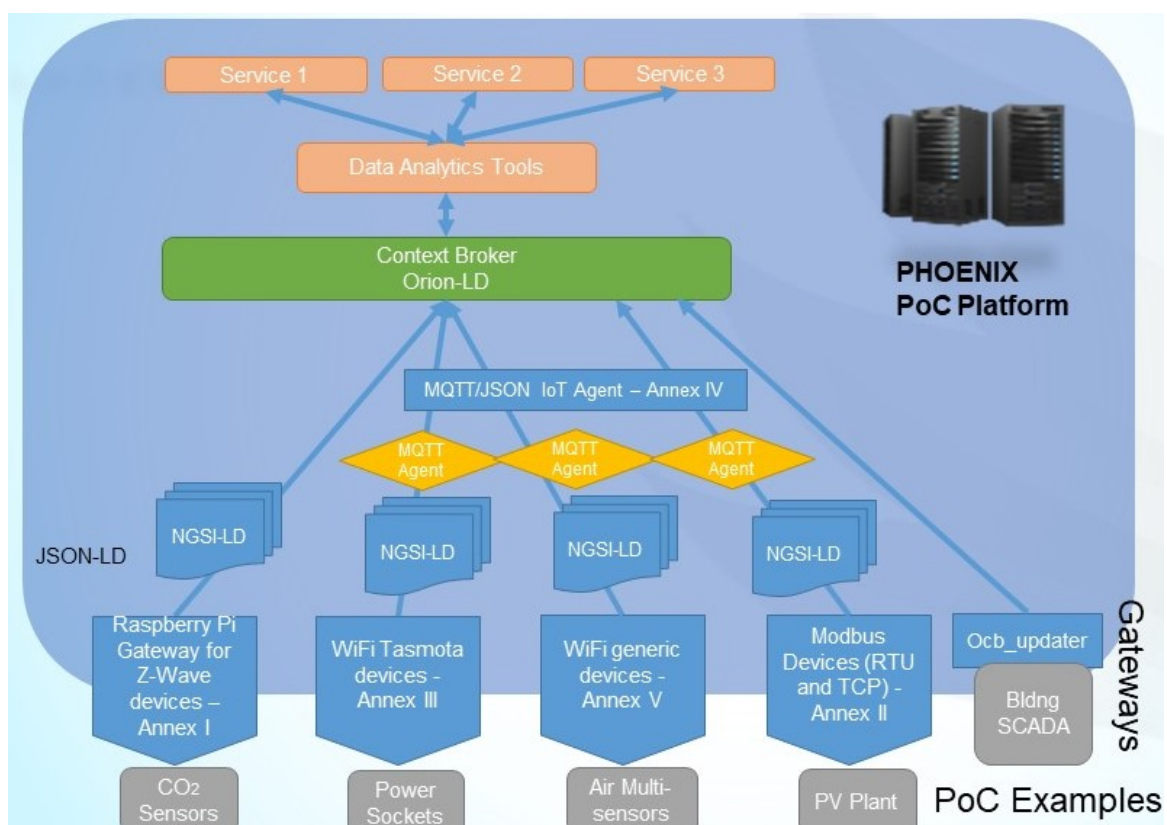
One of the aims of this document is to serve as a guide to perform the transfer of know-how from the PoC to the rest of the partners so they can integrate their systems to the PHOENIX platform. For that, **a series of guides** have been prepared and are part of this document in the form of

annexes. The reason to include this guides as annexes is that it will be easier to disseminate them as stand-alone documents for the rest of the partners and eventually for the rest of the community.

## 7. Design of agents to integrate on the broker

In order to communicate with the legacy devices and building systems which are going to be integrated in the different pilots, there are a number of agents to be configured and tested. Depending on the technology used by each device, it will be necessary to select the type of middleware or gateway.

For the PoC a set of agents have been integrated that allow communication with TCP/RTU connected devices, with Modbus devices, with devices using a Z-WAVE adaptor gateway and with the SCADA of the building using a middleware designed for purpose. This is shown Figure 11.



**Figure 11. Representation of the platform used for the PoC with an emphasis on the connection mechanisms of the legacy equipment.**

### 7.1. FIWARE Smart Data Model



The PoC has carried out the task of defining a data model for the transfer of information based on accepted schemes. With this it was intended to standardise the data formats and to provide value to the higher layers of the platform within the information that is sent. For the purpose of modelling, the smart-data-models of FIWARE was used.

Smart Data Model is a unified approach to data models along with contributions from international organizations such as FIWARE, GSMA and TMForum and IUDX. These data models have been harmonized to enable data portability for different applications including, Smart Cities/Buildings, Smart Agrifood, Smart Environment, Smart Sensoring, Smart Energy, Smart Water and other domains. They are intended to be used in any field but compliance with FIWARE NGSI version 2 and NGSI-LD is necessary.

These data models are available in the following repository <https://github.com/smart-data-models>. The lower level repository is a Subject (i.e. Alert, Streetlighting, etc). Every subject repository is aggregated into domain repositories. Domain repositories compile several subjects. At the same time, a subject could appear in several domains (i.e. Weather appear in Smart Buildings/ Cities). For the PHOENIX data model, the entities of the Building scheme, and entities on the smart cities schemes were used. Also and to complete the definition of the data, a new class was created that was given the name of *Zone* this new class was defined to be contained within a building, and the device was configured to be located within it.

TO make sure that the data streams had enough information relevant for the data analytics part, the QoI schema was used, and the classes *Stream* and *StreamObservation* selected to be considered as the final entity that is updated as the data is produced live by the sensors. The classes used can be seen in Figure 12, and an example can be seen at the end of the deliverable.

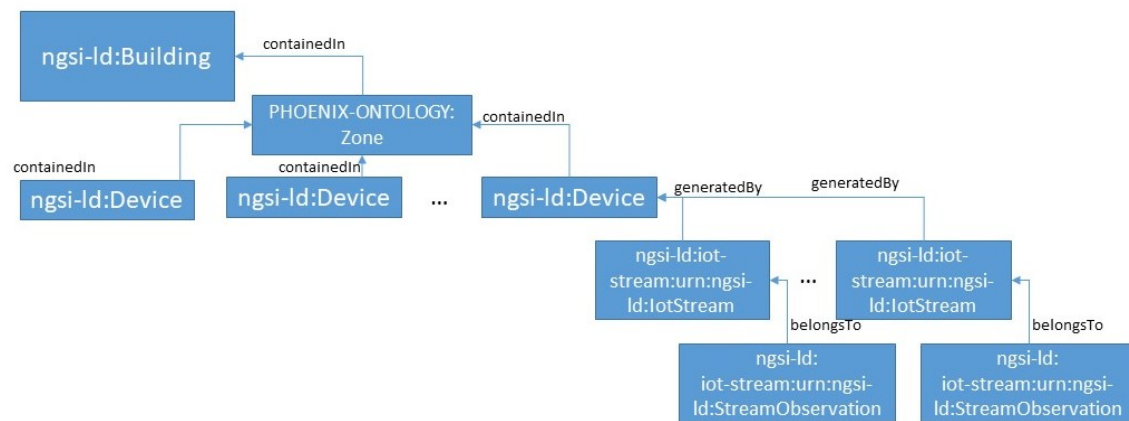


Figure 12. Example of hierarchies in the data model used for the data collected for the PoC.

## 8. Integration plan for the pilots based on PoC learnings

The learnings from the PoC and the study of the situation in each one of the pilots have served to create a plan for the different integrations necessary in each one of the pilots. This was done after a series of meetings of the PoC leaders (UMU) which each one of the pilot managers to evaluate their situation and to agree to options.

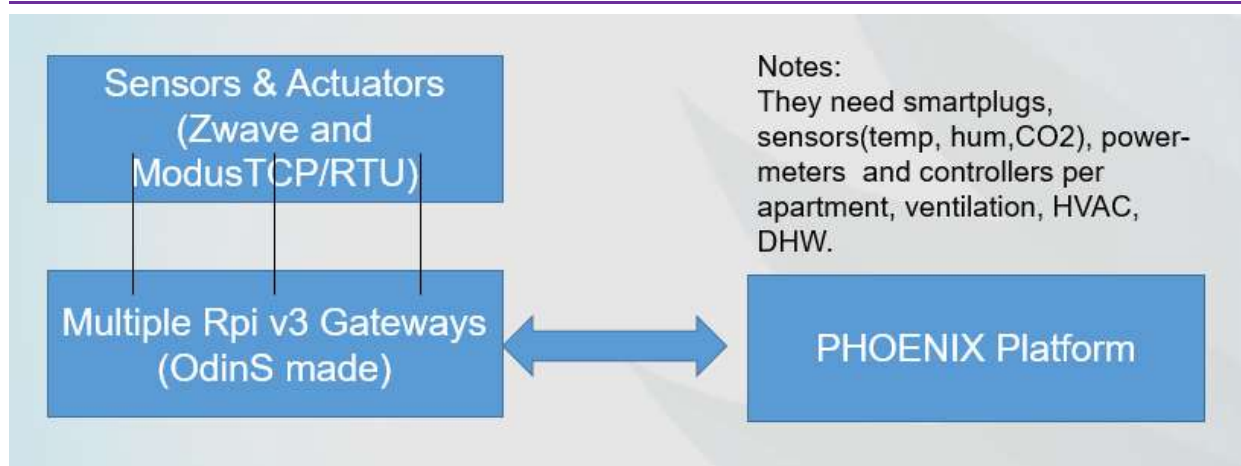
One of the lessons learnt on this respect have been that the buildings normally present a given configuration in which one method / type of device has been used, and the adaptation/integration needs to be done therefore with a given solution. In the following we illustrate with diagrams the plan of action for the different pilots.

### 8.1. UMU

The UMU pilot needs to be considered as a special case as the connectivity effort comes inherited from the PoC. The effort on the UMU pilot will be on increasing the number of devices that have been tried in the PoC, and to enhance the feasibility of the implementation of services at a larger scale, and to transfer the knowledge of the integration to the other partners.

### 8.2. Miwenergía

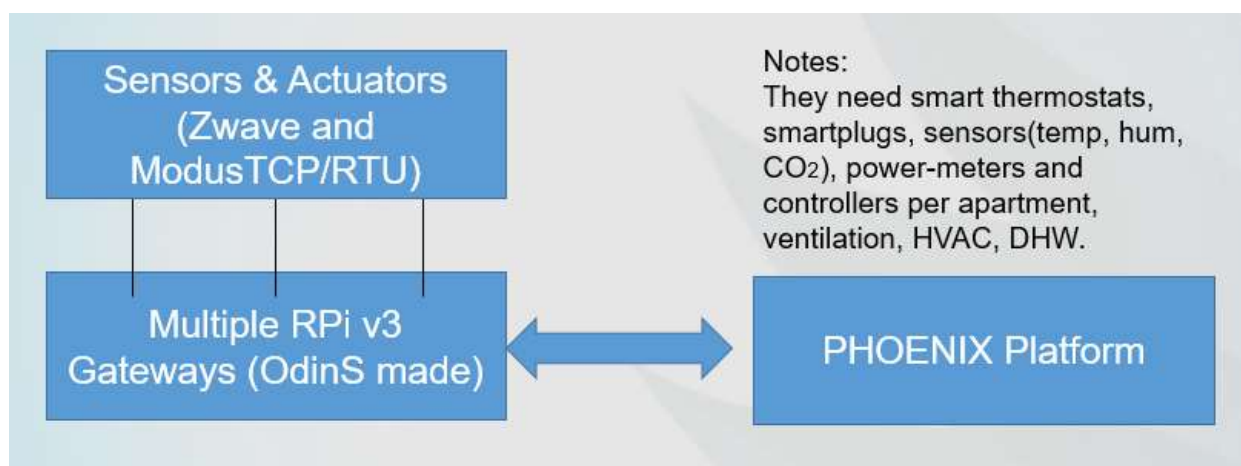
In the case of Miwenergía, the legacy equipment is limited and not connected to the internet. This opens the door to install new hardware that has been selected to be easily connected to the PHOENIX platform using the Z-Wave configuration.



**Figure 13. Overall integration plan of the Miwenergía equipment.**

### 8.3. KaMa

The situation in KaMa is rather similar to the case of Miwenergía. In the KaMa pilot, the only smart devices already installed are smart thermostats that control the refrigerant flow that serve the fan coils to perform the conditioning of the apartments. After study of those devices, it was seen that the integration of them would have required a large amount of resources. As there are easier options to implement to the PHOENIX platform, it was decided to replace the devices with ones that are controllable by gateways as performed on the PoC, and then carry out a configuration effort to establish communication with the PHOENIX platform.

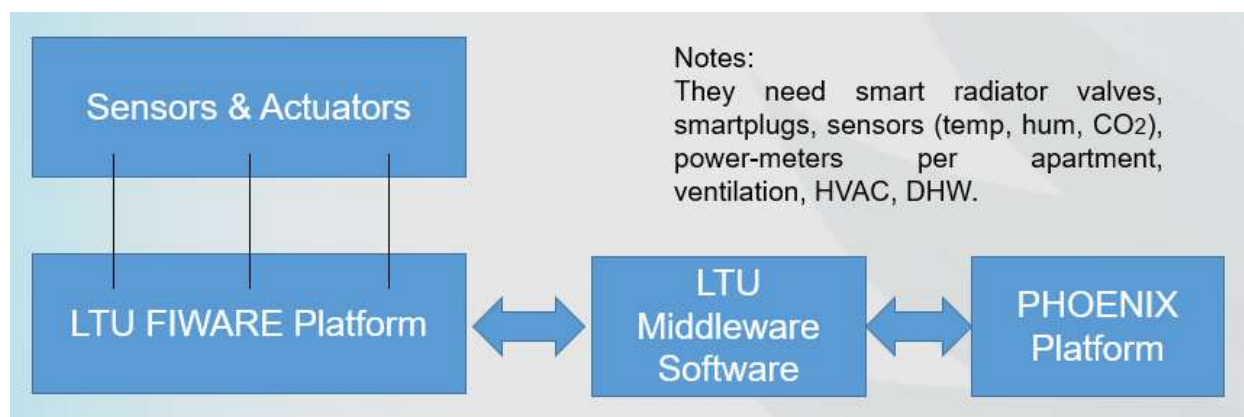


**Figure 14. Overall integration plan of the KaMa equipment.**

### 8.4. LTU

With respect to the LTU pilot, the connection of the sensors and actuators are already in place and connected to a platform managed by LTU. Their platform is a FIWARE one, and the connection with it will be done via a middleware that will be created by LTU for this specific purpose. The

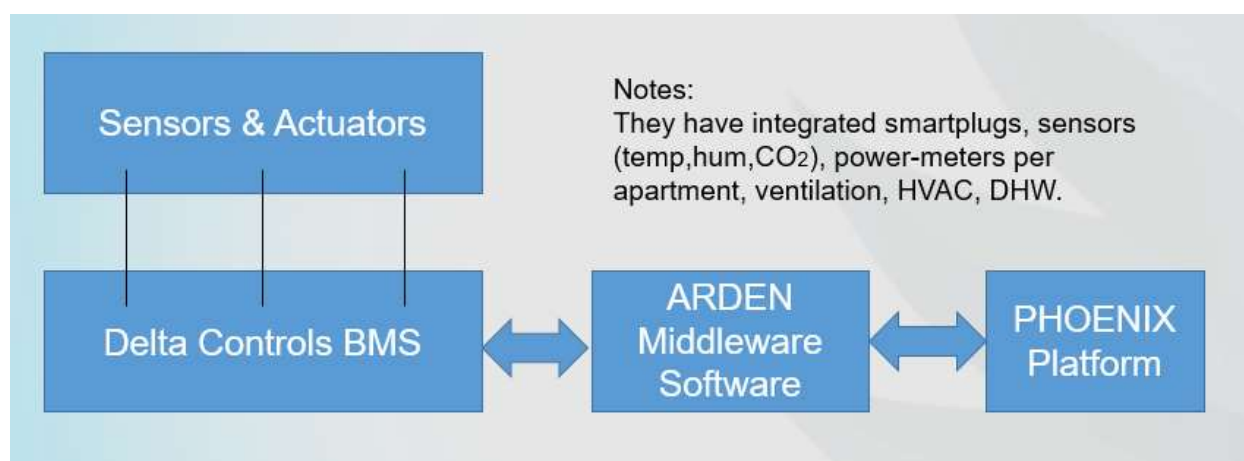
fact that both platforms use the same language has facilitated the development of the software.



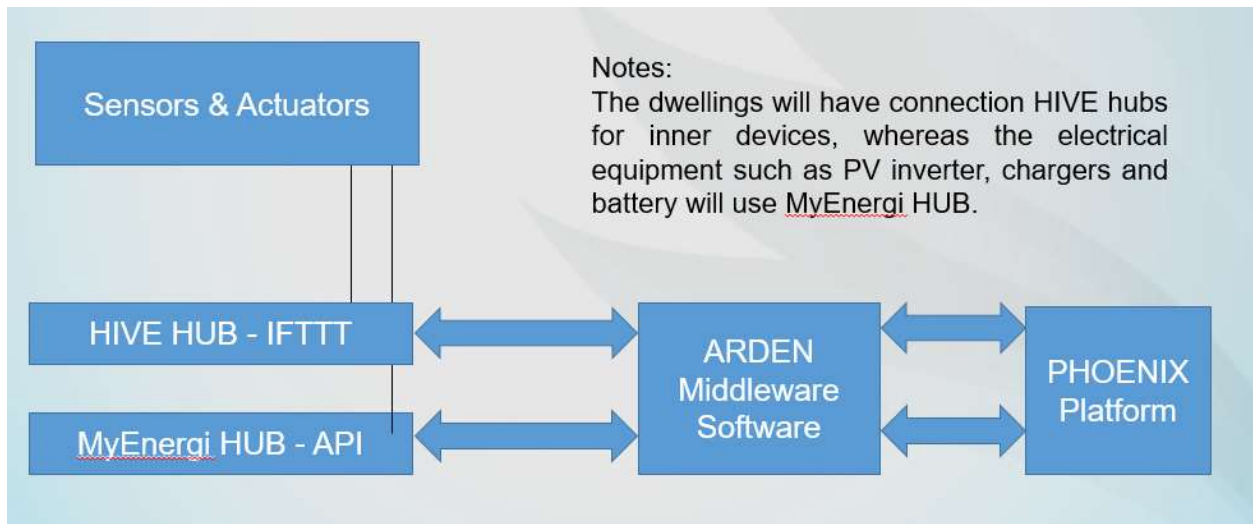
**Figure 15. Overall integration plan of the LTU equipment.**

### 8.5. ARDEN

The ARDEN pilot is formed by two different sites, both have a large wealth of data which is locked at the moment on closed systems that need to be integrated. In the case of the commercial building, the sensors and actuators are connected to a Building Management System that allows control of services. For the connection of this BMS to the PHOENIX platform, ARDEN is going to implement a middleware that is capable to connect the Delta Controls BMS to the PHOENIX platform, with this, it will be possible to connect to the sensors and actuators already existing on the commercial building.



**Figure 16. Overall integration plan of the ARDEN equipment on the commercial building.**



**Figure 17. Overall integration plan of the ARDEN equipment on the domestic building.**

With respect to the domestic sites, the ARDEN pilot will install new hardware that will be selected with the purpose of easing the connectivity to the PHOENIX platform. The HIVE and MyEnergi solutions will be used and their manufacturers have been contacted to verify if they offer connectivity options. An API will be provided by both manufacturers to enable the connection of the hubs with the PHOENIX platform.

## 9. Conclusions and further work

This document provides technical information and guidelines about the hardware/software integrations developed in the PHOENIX project to support the implementation of the Proof-of-Concept pilot in the Murcia premises.

For the development of this work, a great deal of investigation about the SRI and the status of the UMU site had to be done in order to (1) select the equipment that was more relevant to be integrated onto the PHOENIX platform and (2) design a plan of action to make that legacy equipment communicate with the PHOENIX platform.

As for conclusions, it is noted that legacy building devices in existing buildings, at least in the area in which the PoC is located, generally lack sufficient connectivity to turn a non-connected building into a smart building. Although some manufacturers are starting to offer the possibility of adding communication cards to their systems, this will be only interesting if the whole building has a monitoring and sensing plan.

Notwithstanding that, the PoC demonstrates the feasibility of incorporating of those systems on a

platform when the platform is in established with connectivity protocols. Larger equipment, such as conditioning systems or Air Handling Units, appear to be the most advanced in terms of connectivity and interoperability, and at the same time, they are the larger consumers of the building, so their connection provides significant opportunities for peak shifting and energy saving.

With respect to sensors on the indoor environment, the recent pandemic of COVID-19 has stimulated an interest in building users on the air quality of the environments surrounding them. This has led to the installation of air quality sensors, or the petition of the workers on having one, particularly CO<sub>2</sub>, as this has been the recommendation by some governments. The PoC always planned on integrating CO<sub>2</sub> sensors as they are known to fundamental indicators of the air quality of buildings. This has fitted well with the requirements of the users, and it opens the door to the integration for services that will be aimed at controlling the propagation of airborne pathogens such as SARS-COV-2.

As future works, the integration techniques explained here and planned within Section 8 will be replicated and extended in the remaining pilot scenarios developed in the WP7 of the PHOENIX project.

## 10. Bibliography

How-to Raspberry Pi Wifi configuration: <https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md>

Country Codes according to ISO 3166-1: [https://en.wikipedia.org/wiki/ISO\\_3166-1#Current\\_codes](https://en.wikipedia.org/wiki/ISO_3166-1#Current_codes)

Tasmota firmware Power Monitoring Calibration: <https://tasmota.github.io/docs/Power-Monitoring-Calibration/>

Stijn Verbeke, Dorien Aerts, Glenn Rynders, Yixiao Ma, Paul Waide. 3<sup>rd</sup> interim report of the 2<sup>nd</sup> technical support study on the smart readiness indicator for buildings. Vito NV, Belgium.



## Example JSON-LD data model

```
[
  {
    "id": "urn:ngsi-Id:Device:UMU-
Pleiades-BlockB-B1.1.014-CO2",
    "type": "Device",
    "category": {
      "type": "Property",
      "value": [
        "sensor"
      ]
    },
    "description": {
      "type": "Property",
      "value": "CO2 sensor for room
B1.1.014"
    },
    "containedIn": {
      "type": "Relationship",
      "object":
"urn:PhoenixOntology:Zone:UMU-
Pleiades-BlockB-B1.1.014"
    },
    "location": {
      "type":
"GeoProperty",
      "value": {
        "type": "Point",
        "coordinates": [
          38.02418,
          -1.17311,
          630
        ]
      }
    },
    "controlledProperty": {
      "type": "Property",
      "value": [
        "co2"
      ]
    },
    "dateLastValueReported": {
```

```
    "type": "Property",
    "value": {
      "@type": "DateTime",
      "@value": "2021-04-
28T11:00:00Z"
    },
    "@context": [
      "https://schema.lab.fiware.
org/ld/context",
      "https://phoenix-
project.eu/data-
models/context.jsonld"
    ],
    {
      "id": "urn:ngsi-
Id:IoTStream:UMU-Pleiades-BlockB-
B1.1.014-CO2",
      "type": "IoTStream:IoTStream",
      "IoTStream:generatedBy": {
        "type": "Relationship",
        "object": "urn:ngsi-
Id:Device:UMU-Pleiades-BlockB-
B1.1.014-CO2"
      },
      "@context": [
        "https://uri.etsi.org/ngsi-
Id/v1/ngsi-Id-core-context.jsonld",
        {
          "IoTStream":
"http://purl.org/iot/ontology/iot-
stream"
        }
      ],
      {
        "id": "urn:ngsi-
Id:Observation:UMU-Pleiades-
BlockB-B1.1.014-CO2",
        "type": "IoT-
stream:StreamObservation",
        "IoTStream:belongsTo": {
          "type": "Relationship",
          "object": "urn:ngsi-
```

```
Id:IoTStream:UMU-Pleiades-BlockB-
B1.1.014-CO2"
      },
      "hasSimpleResult": {
        "type": "Property",
        "value": "450"
      },
      "observedProperty": {
        "type": "Relationship",
        "object": "urn:ngsi-
Id:controlledProperty:co2"
      },
      "@context": [
        "https://uri.etsi.org/ngsi-
Id/v1/ngsi-Id-core-context.jsonld",
        {
          "IoTStream":
"http://purl.org/iot/ontology/iot-
stream",
          "hasSimpleResult":
"http://www.w3.org/ns/sosa/hasSimpl
eResult",
          "observedProperty":
"http://www.w3.org/ns/sosa/observed
Property"
        }
      ]
    }
  ]
}
```



## Annex I - Raspberry Pi Gateway for Z-Wave devices

Some of the pilots are going to use a Raspberry Pi (or an extended version of it) as the IoT Gateway. The instructions to be followed take for granted that the Raspberry Pi is using Raspian or Raspberry Pi OS.

### WiFi configuration

The easiest way to connect the Pi to an existing network is by using its WiFi interface that has to be configured. Remember to configure all the country-related options in **5 Localisation Options** if you haven't done this before configuring the WiFi (see how execute *raspi-config* in the next paragraph).

The preferred way to do that is by using the *raspi-config* utility (some sections might be slightly different depending on the version of the OS that's being used). Run it with:

### *sudo raspi-config*

By default you must go to **1 System Options** → **S1 Wireless LAN** and set there both SSID and passphrase.

Another option is to configure it manually. In this case you will need to modify the file */etc/wpa\_supplicant/wpa\_supplicant.conf* which includes some information that **must be at the top of the file** since the last version of *Buster Raspberry Pi OS* (see the example below, the 3 first lines) and is followed by the configuration of the network itself (SSID, etc.).

If the network SSID is visible and the network isn't open, use this:

```
wpa_passphrase "YOUR_NETWORK" | sudo tee -a /etc/wpa_supplicant/wpa_supplicant.conf  
>> /dev/null
```

You will be asked to introduce the passphrase (in the next example it's *testingPassword*).

Write *cat /etc/wpa\_supplicant/wpa\_supplicant.conf* to see the contents of the file:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=XX
network={
    ssid="YOUR_NETWORK"
    #psk="testingPassword"
    psk=131e1e221f6e06e3911a2d11ff2fac9182665c004de85300f9cac208a6a80531
}
```

The country code *XX* is the two-digit version according to ISO 3166-1.

In the network section you can see the passphrase in both plain text (the commented line, which should be removed for security reasons) and encoded format (a part of it is generated randomly). If you intend to modify the network section, remove it in the file before running the *wpa\_passphrase* command again.

As terminal text editor, you can use:

```
sudo nano cat /etc/wpa_supplicant/wpa_supplicant.conf
```

Just change what you need and press **Ctrl+X**, then **Y** and finally **ENTER**.

If you're going to use an open WiFi network, the network section should look like this:

```
network={
    ssid=" YOUR_NETWORK "
    key_mgmt=NONE
}
```

**Note (important!):**

If you're going to use DHCP in order to get a dynamic IP address (default), in the background, there will be a daemon that's in charge of controlling all this process called *dhcpcd*. Part of its configuration is stored in the folder */var/lib/dhcpcd5* and includes the file *duid* (*dynamic unique*

*identifier*). As its name says, this identifier ***must be unique*** and if you're cloning a SD image, you will get the same value for multiple IoT Gateways. Considering that the daemon will create one if the file doesn't exist, remove the contents of the folder just this first time as a way to start from scratch with:

```
cd /var/lib/dhcpd5 && sudo rm -f *
```

Once you're done, you can apply the changes by restarting the device with:

```
sudo reboot
```

### **Wireless Z-Wave Integration by Raspberry PI gateway**

Z-Wave is a Wireless communication protocol whose primary area of use is home automation. It is capable of establishing a mesh network, meaning that it can use end devices as intermediaries for communications, successfully expanding the range of the network beyond the radio-frequency coverage of the main gateway.

Among the devices that we can find, using Z-Wave as its communications technology, we can find light switches, smart hubs, thermostats, smart plugs, CO<sub>2</sub>, temperature, humidity and many other sensors, to name a few.

The integration with Z-Wave devices is also done using a Raspberry Pi but in this case a Z-Wave gateway must be used to communicate with the devices.

In the PoC, in the *Air Quality monitoring using CO<sub>2</sub>* system, an Aeotec USB Z-Stick Gen5 was used.

The first step to be followed is pairing the Stick and the devices. To do that, you must press the Stick's button (it's surrounded by a led circle that's off by default) just one time (short press) and you'll see the led blinking slowly. This means it's in pair mode.

While it's in this mode, put the Stick next to the device (you can unplug it from the USB connector,

it has a small battery inside) and push the device's pairing button. Wait for 3 seconds and then push the Stick's button once again to go back to normal mode (led off). If everything works fine, both should be paired.

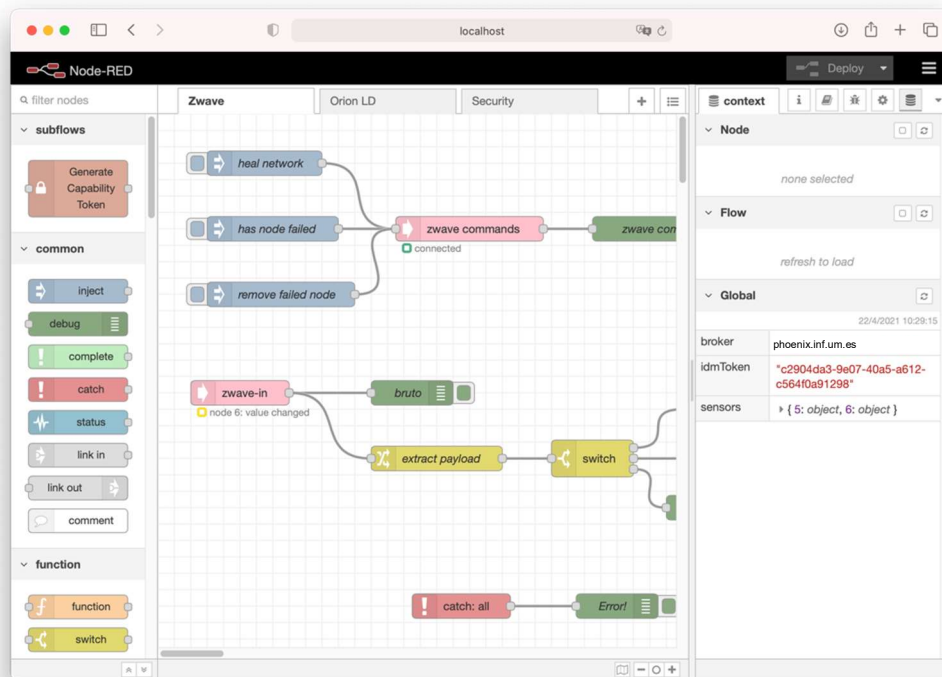
You can also *unpair* a device from the Stick and the process is pretty much the same, but in this case you must switch the Stick to *unpair* mode with a long press (about 3 seconds) and while it's in this mode, you'll see the led blinking faster. Other than that, everything's the same as in the pairing process.

**Note (important!):** If you're going to use the Aeotec USB Z-Stick Gen5, you can't use a Raspberry Pi 4 due to some compatibility issues. Instead use a Raspberry Pi 3B+.

### **Firmware of Raspberry PI for Z-Wave integration**

In order to be able to interface the wireless Z-Wave devices and integrate the information into the *Phoenix Platform*, we have developed a custom firmware in Raspberry PI based on the *Node-RED* platform.

*Node-RED* (as featured in Figure 18) is a flow-based visual programming development tool. It was originally developed by IBM for wiring together hardware devices, online services and APIs in the Internet of Things.

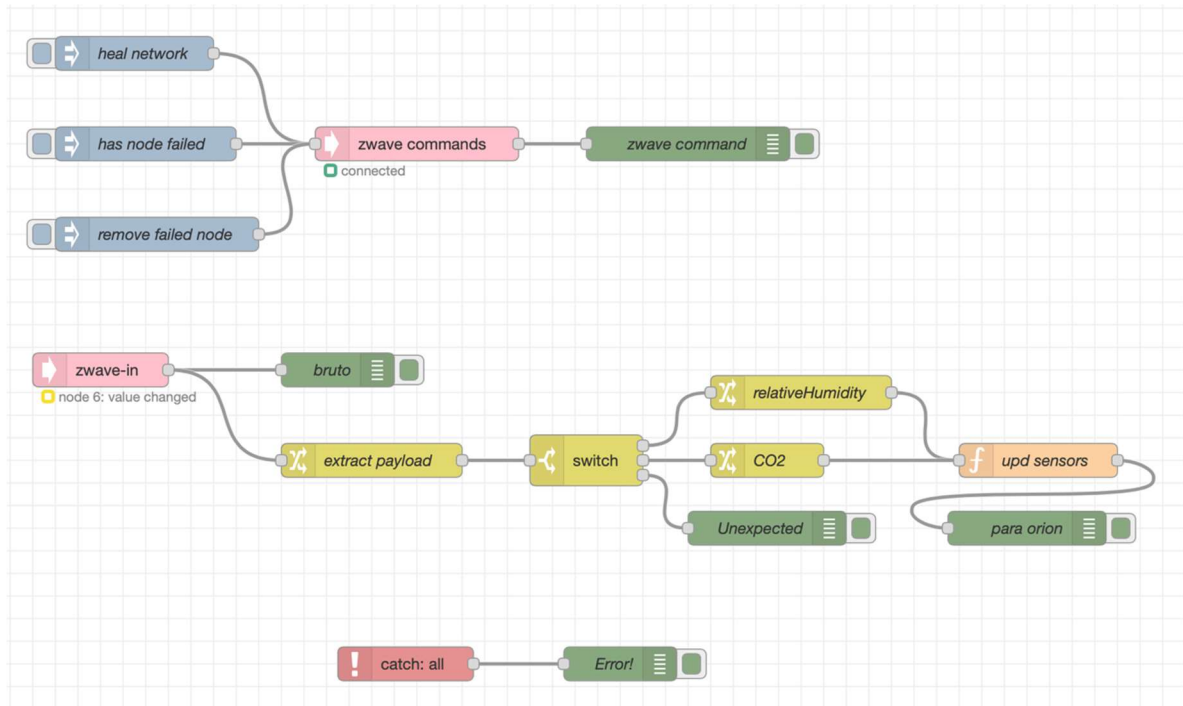


**Figure 18 Node-RED programming GUI**

This development environment allows the programmer to intuitively connect different nodes in a processing flow. Different nodes act as sources, intermediary or processing elements or sinks. Running over *Node-JS*, it has good performance over constrained hardware, such as the Raspberry Pis on which the current gateways are based. In order to be able to interface with Z-Wave Gateway sticks, *Node-RED* utilizes the *Open-Z-Wave* library, which has to be installed prior to the use of the Z-Wave-related nodes.

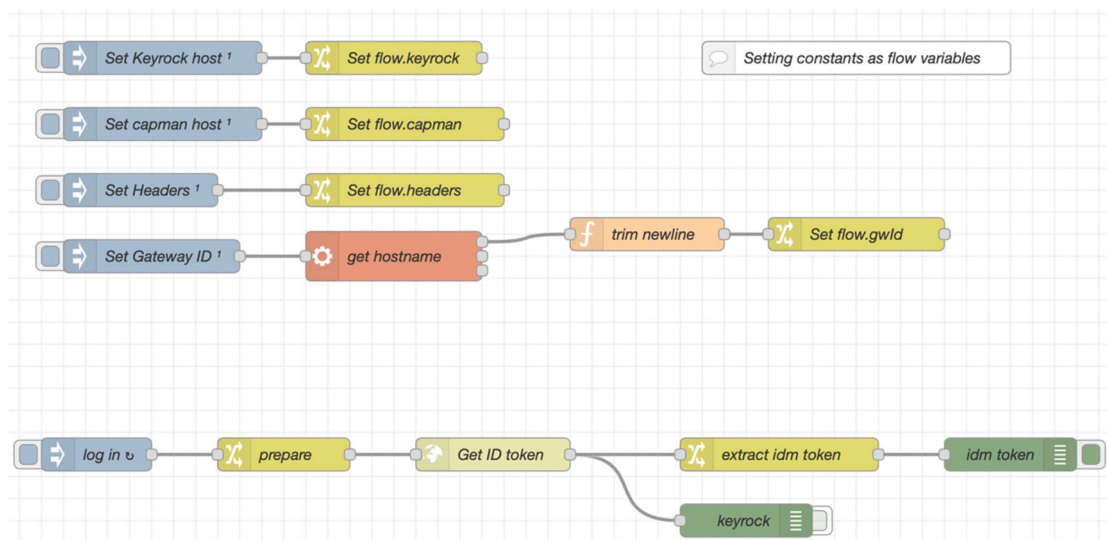
In order to integrate Z-Wave devices with *Phoenix Platform*, three “flows” have been developed, one for gathering all the information from the Z-Wave devices, another one for managing the security interactions and finally one to manage context actualizations with the broker.

In Figure 19, the different processing flows of the Z-Wave information retrieval can be seen. In it we can appreciate how the “Zwave-in” node acts as the source of Z-Wave notifications coming from the devices. Those notifications carry any changes in state perceived by the different devices, such as changes in perceived CO2 or temperature. That information is later processed, extracting interesting bits and further processing the filtered information, finally updating the global repository of context information shared among flows, in the “upd sensors” node.



**Figure 19 Z-Wave interactions in Node-RED**

In Figure 20, a similar flow is presented, this time for the management of the security components of the *Phoenix Platform*. In it we obtain the identity token from the IDM, which will allow us in turn to retrieve specific capability tokens from the Capability Manager. The capability tokens will later be used in order to interact with the broker, thus they are stored in the global information shared among flows, in a similar manner than in the previous flow.



**Figure 20 IDM and Capability Token interactions in Node-RED**

Finally Figure 21 shows the interactions with the broker. This flow performs periodic updates on the information held by the context broker, representing the different Z-Wave devices under the

control of the Z-Wave Gateway. As we saw in previous flows, the context information representing device states, as well as the capability tokens, are stored and retrieved from the global configuration repository held by Node-RED, which is updated by the corresponding flows.

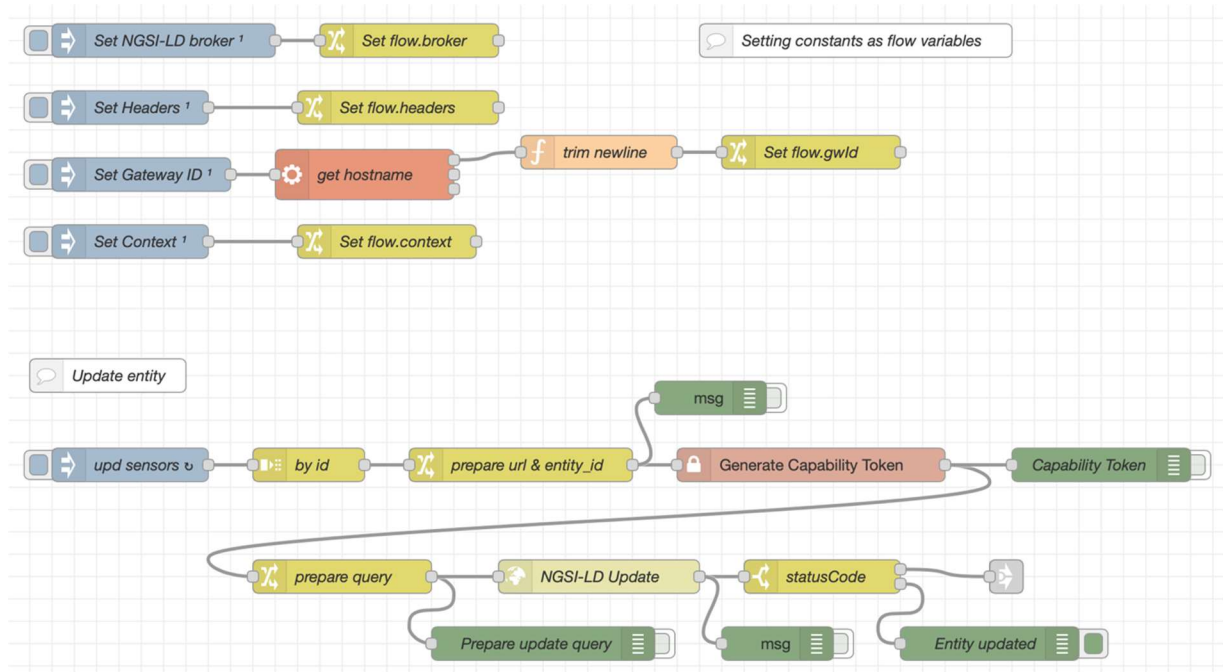


Figure 21 Broker interactions in Node-RED

## Integration of new Z-Wave devices

Any new device which has to be connected to the Z-Wave gateway, has to follow the usual Z-Wave linkage procedure. It usually is automatically performed by the device on the first run: searching for available gateways in the proximity and then linking to them. In case a rebinding must be performed, follow the instructions of the manufacturer, both of the Z-Wave Gateway stick to unregister any removed devices and register new ones, as the Z-Wave device, to rebind with a new Z-Wave gateway. Once a device is paired with a Gateway stick

Each device will be represented by a different entity in the broker, with its associated identifier and type descriptors. The entity id is formed based on the unique identifier of the Raspberry Pi that collects the data from the device (which is manually configured on each Raspberry Pi), as well as the unique identifier of the device within the scope of the Z-Wave Gateway stick group of devices.

The Node-RED flows have been prepared for some example devices, but as new devices are continuously being created, there could show up the need to integrate a new model which is not readily covered by the existing flows. In order to do so, we can utilize *Node-RED* in-built debug

panels and development helps, in order to filter the information of the newly integrated device in the Z-Wave flow, and adapt its information to the internal context management which will be periodically updated by the broker flow.



## Annex II - Modbus Devices (RTU and TCP)

In the pilots where some devices are going to be accessed using the Modbus protocol, they must be tested first to make sure they work as expected and that the documentation available regarding the register map is correct.

For performing this set up, it is needed an application that can play the role of a Modbus Master and can work in both RTU and TCP mode. *Linux* users can use *QModMaster* but they'll probably have to compile it from sources, or a Windows alternative can be chosen.

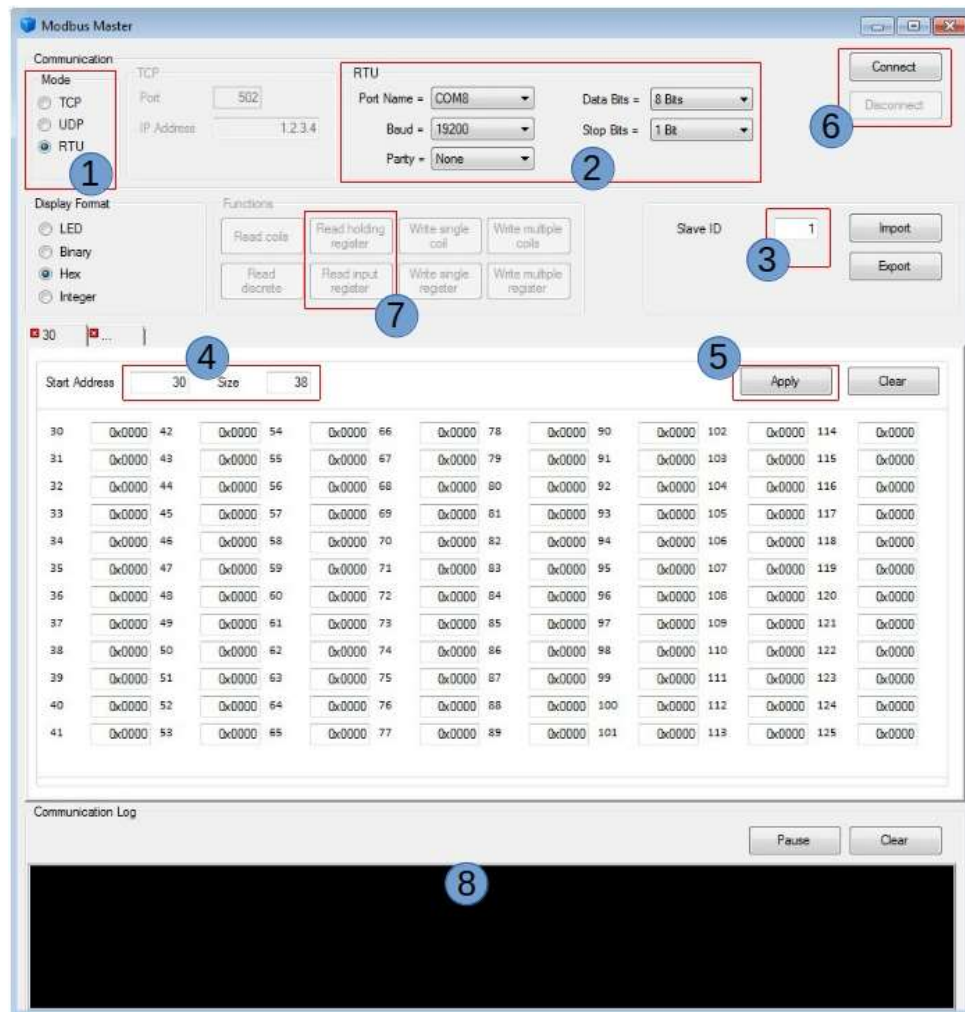
Whether the device uses RTU or TCP mode (first during the test and next during the configuration phases), the Modbus register map ***must be available***. It doesn't mean that all the registers have to be used though.

### Modbus RTU mode

In RTU mode, the physical interface is normally a RS-485 bus. The fastest way to test it is by using an USB ↔ RS-485 converter and there are many manufacturers for this type of devices using different chipsets (FTDI is one of them).

Once the drivers are installed (they might be already installed by default depending on the version of the OS used during the tests), a new serial port will appear in the *Device Manager*.

Locate it and run the app:

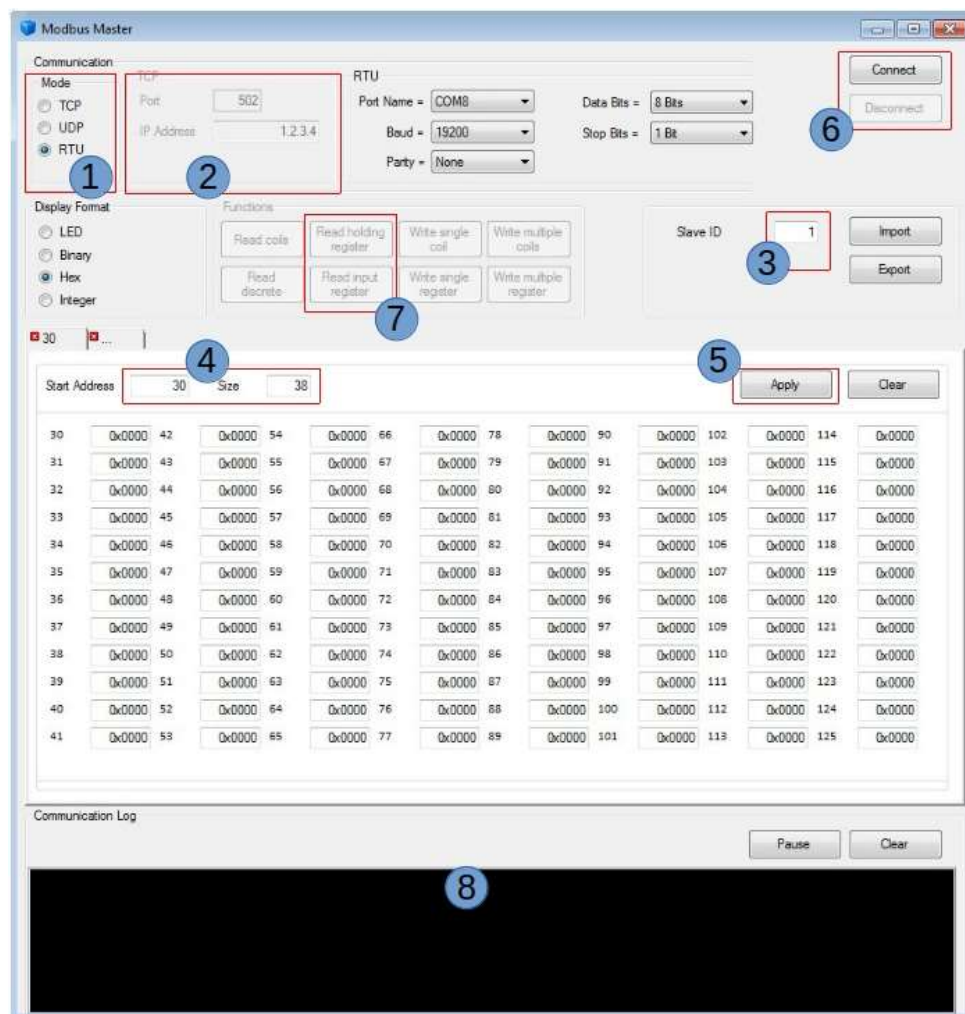


The steps to test the communication are:

1. Select RTU mode.
2. Select the serial port created by the adapter and configure it according to the proper values indicated by the datasheets of the slave devices.
3. Set the slave address.
4. Choose the offset of the first register (Start Address) and the number of registers (Size). Both values are always integer.
5. Click on Apply after that.
6. Connect.
7. Send the read request (Read Holding Register = Function 3 and Read Input Register = Function 4). The most common Function is Read Holding Register.
8. Check the result of the operation and, if everything worked fine, the data will have been updated in the main panel. By default, the values are represented in hexadecimal but they

9. Optionally you can modify the values if supported by the device (Write Single Register = Function 6 and Write Multiple Register = Function 16).

In this mode the converter is not necessary. Here the slave provides a TCP socket that is listening for incoming connections by default on port 502.



Keep in mind that slaves sometimes close the connection if there's no activity for some time (usually from 30 to 60 seconds). So during the tests, unless you're sure that you're exchanging

data quite often, you should better *Disconnect* and *Connect* again. Otherwise the app may end up getting frozen waiting for a response from the device that's not going to come.

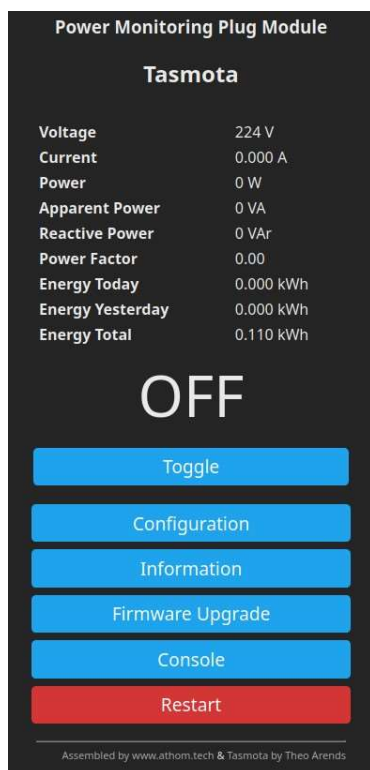
## Annex III - WiFi Tasmota devices Example of Smart Sockets

The sockets used in the PoC are a version that come pre-programmed with the tasmota firmware. This guide shows how to integrate them on the PHOENIX platform using the MQTT agent.

### Smart Socket Configuration

#### WiFi configuration

The first step to follow in order to configure the device is to enable its internal WiFi network. To do this, push the button 6 times fast (short press 6 times) and the LED of the button should start blinking slowly indicating that its WiFi is on. **Do not run a long press as it will remove current configuration and calibration values!**



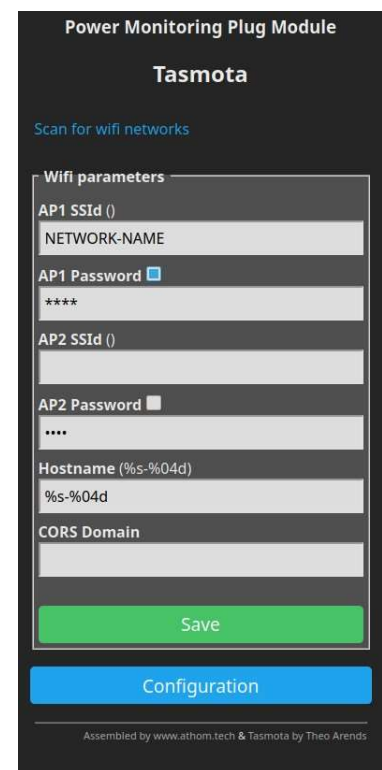
With a laptop or a smart-phone you should see an active WiFi network named *tasmotaXXX* that has no internet connectivity. Join the network and open the next URL in a web browser:

<http://192.168.4.1>

First of all, go to the *Information* section and obtain the *MAC Address* (you may need it).

You can go now to *Configuration* → *Configure WiFi* to set the proper

values for the network the device is going to connect to.



Just configure the *SSID* and the *Password* for AP1 and let empty the *SSID* for AP2. You can then *Save* the configuration. Note that the device will go back to normal operation mode if there's no activity for some time, so if that's the case, you will have to go to your DHCP server's status page (normally in your router) to get the IP address assigned to the device (here is where you'll have to

use the *MAC Address*).

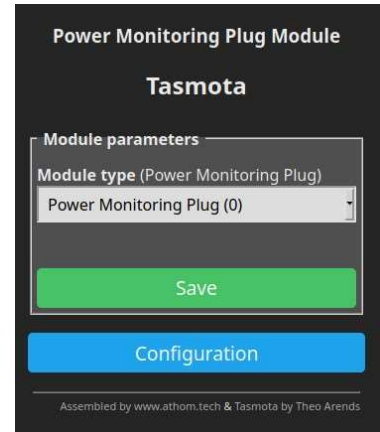
## Module type configuration

For the configuration of the module, the user has to go to the section of *Configuration* → *Configure Module*

Choose as *Module Type* the first option in the combo, *Power Monitoring Plug (0)*.

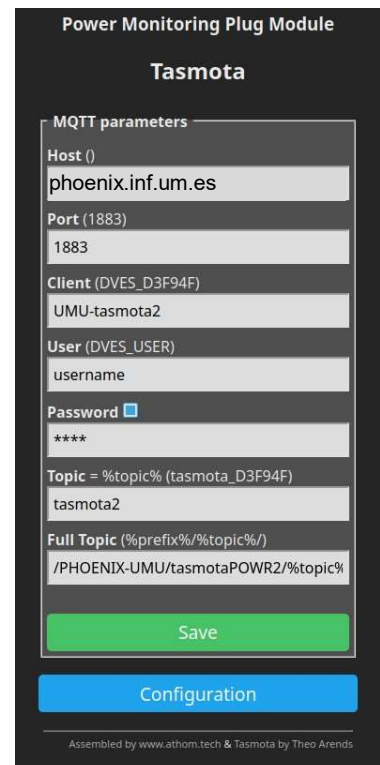
## MQTT connection configuration

Go to section *Configuration* → *Configure MQTT*



Fields to configure in this section:

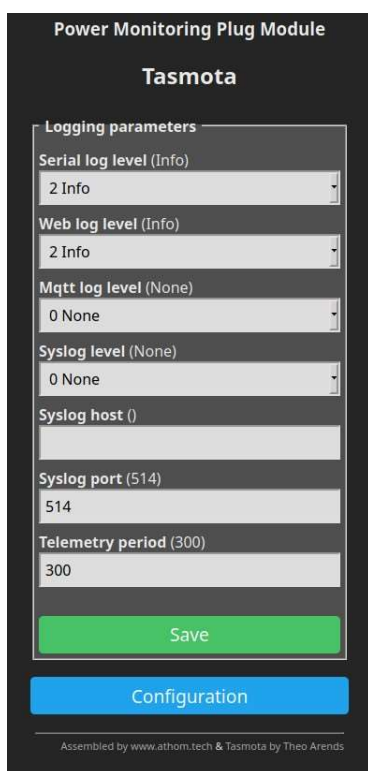
- Host. The MQTT broker is listening on phoenix.inf.um.es.
- Port. The MQTT broker is listening on port 1883 (this is the default MQTT port).
- Client (*unique Client ID* for the device in the broker). Use **PILOTNAME-** as prefix and add a unique suffix per device (use the pattern *tasmotaX* being X a number starting from 1).
- User. Username/Password for the connections will be sent separately for each partner as there have been established some security restrictions in order to prevent collisions between devices from different pilots. Each pair User/Password can be used for all the tasmota devices of the same pilot.
- Password. Password for the connection (the checkbox must be selected).
- Topic. Use the same value as the suffix added at the *Client ID* (that is, *tasmotaX*).
- Full Topic. Full topic that combines a specific *API-KEY* for the pilot (**PHOENIX-PILOTNAME**), a second level common to all this type of devices (*tasmotaPOWR2*) and a third level related to the *Topic* (*%topic%*). All combined: **/PHOENIX-PILOTNAME/tasmotaPOWR2/%topic%**



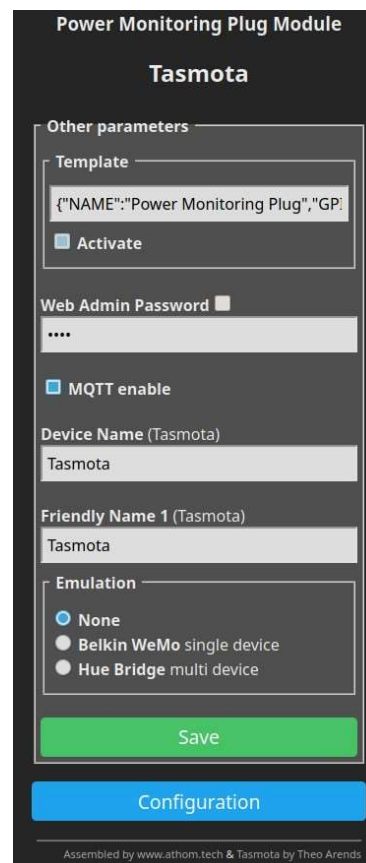
## Logging configuration

Go to section *Configuration* → *Configure Logging*

In this case you just need to configure the measurement period (*Telemetry period*, in seconds) and a reasonable value could be 5 minutes between readings (300 seconds). Keep in mind that the readings are bound not to be aligned in time as this parameter is only indicating how often the information is sent. So don't expect to get readings exactly in minutes 0, 5, 10 and so on.



The screenshot shows the 'Logging parameters' section of the Tasmota web interface. It includes dropdown menus for 'Serial log level' (set to 2 Info), 'Web log level' (set to 2 Info), 'Mqtt log level' (set to 0 None), and 'Syslog level' (set to 0 None). There is an empty text field for 'Syslog host' and a text field for 'Syslog port' (set to 514). The 'Telemetry period' is set to 300. A green 'Save' button is at the bottom of the section, and a blue 'Configuration' button is below it. The footer text reads 'Assembled by www.athom.tech & Tasmota by Theo Arends'.



The screenshot shows the 'Other parameters' section of the Tasmota web interface. It includes a 'Template' dropdown menu with the value '{\"NAME\":\"Power Monitoring Plug\",\"GP'. Below it is an 'Activate' checkbox. The 'Web Admin Password' field is masked with four dots. The 'MQTT enable' checkbox is checked. The 'Device Name' and 'Friendly Name 1' fields both contain the text 'Tasmota'. The 'Emulation' section has three radio buttons: 'None' (selected), 'Belkin WeMo single device', and 'Hue Bridge multi device'. A green 'Save' button is at the bottom of the section, and a blue 'Configuration' button is below it. The footer text reads 'Assembled by www.athom.tech & Tasmota by Theo Arends'.

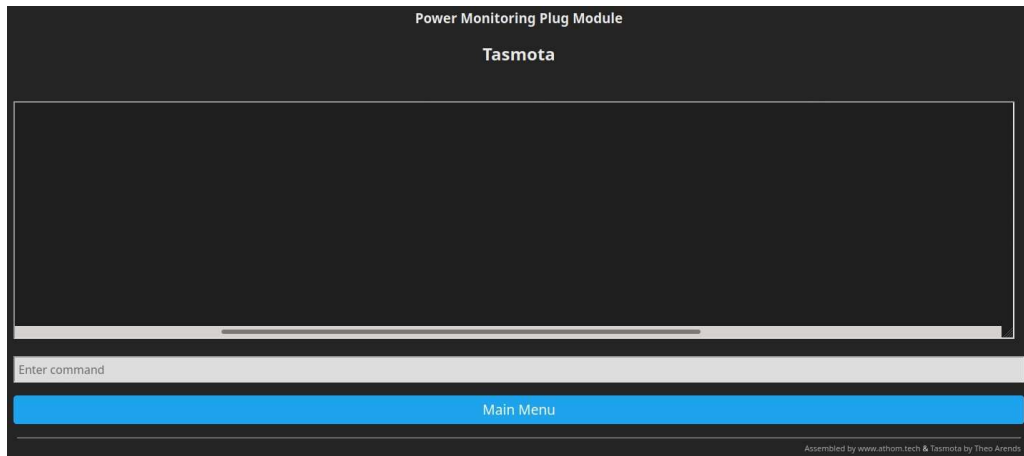
## Other aspects to configure

Go to section *Configuration* → *Configure Other*

Make sure here to keep selected the *MQTT enable* checkbox.

## Calibration of Power Measurement in Smart Socket

Go to section *Console*



The detailed instructions to calibrate the power meter are available here [3].

You need an AC-capable calibrated multi-meter and a known-wattage load **with a power factor as close to 1 as possible** (e.g., a 60W incandescent light bulb).

The steps to follow are these:

- Plug the socket and turn it on (when you push the button, the led goes *Red* when is on and *Blue* when is off).
- Read the *Voltage* with the multi-meter (let's say 227.5V).
- Plug the light bulb in the socket and wait for a few seconds for the reading to stabilize.
- Set the *Power* to 60W in the command line writing **PowerSet 60.0** (press Enter).
- Set the *Voltage* to 227.5V in the command line write **VoltageSet 227.5** (press Enter).
- Set the Current to 263.736mA in the command line write **CurrentSet 263.736** (press Enter).

For the last operation note that  $I(A) = P(W) / V(V)$  and **must be set in mA** (I = Current, P = Power and V = Tension/Voltage). So the full equation is:

$$I(mA) = 1000 * (60 / 227.5) = 263.736$$



## Annex IV - Interoperable APIs provided by the PHOENIX platform

The connection point between the pilots (devices, BMS, external data sources, etc.) and the backend of the *Phoenix Platform* is formed by a group of IoT Agents. Each agent provides a specific format that allows bidirectional communication (actuation only if it's supported by the devices).

### MQTT/JSON IoT Agent

MQTT is a publish/subscribe protocol based on TCP that uses the concept of *topic*. Every message must be sent (published) to a topic and all the clients subscribed to this topic will receive it.

This agent exchanges information in *json* format on top of MQTT messages and the name of the topics is divided in 3 parts:

- API-KEY. It plays the role of a group ID. There will be one API-KEY per pilot.
- DEVICE-ID. This is the unique identifier of the device.
- Topic.

There are 3 well-known topics:

- */API-KEY/DEVICE-ID/attrs* is the topic used by the devices to publish readings.
- */API-KEY/DEVICE-ID/cmd* is the topic used by the agent to publish commands. The device must be subscribed to its own *cmd* topic.
- */API-KEY/DEVICE-ID/cmdexe* is the topic used by the device to confirm the execution of commands.

An example could be a multi-sensor which provides temperature, humidity and CO<sub>2</sub> using the attributes *temp*, *hum* and *co2*. An update of the values could be a publication sent by the sensor to the topic */PHOENIX-UMU/device1/attrs* with the next payload (note that there's no need to update all the attributes in every publish message):

```
{"temp": "25.3", "hum": "47.2", "co2": "450"}
```

Depending on the definition of the attributes, the value of each one of them will have to be sent in numeric or in String format (as in the example above). Once the devices are provisioned, the right format could be established, as default the String format is the preferred one but both should work. Regarding actuation, we could have for example an *on/off* device (let's say a light). A way to switch it on could be done defining the *light1on* command that would be executed sending the next message to the topic */PHOENIX-UMU/light1/cmd*:

```
{“lightIon”:{}}
```

Each command must be unique and there can’t be any attribute with the exact same name as the command, which means there can’t be a *lightIon* attribute. Other than that, and regardless of some minor restrictions (spaces, non-standard characters, etc.), any name could be used for the command (in the previous example it could have been *IwantYOUtoSWITHlightIONnow*).

Other than that, the command is pretty much a json attribute but **must be a json-object** and can include parameters inside the internal *object*. The previous command has no parameter and that’s why the internal *object* is empty.

Once the command is executed, the device must publish a response in the confirmation topic, that is, */PHOENIX-UMU/light1/cmdexe*, with the next content:

```
{“lightIon”:“”}
```

The content of the String value is not defined by the agent and will just be forwarded to the upper layers of the platform. The key point here is that, once the confirmation is received, the agent will finish the execution of the command.

#### Note:

Right now the */API-KEY/DEVICE-ID/attrs* topic must only be used to send real-time values. The mechanism to send historical values has yet to be defined.

### MQTT Examples for communicating with the PHOENIX platform

Be careful with quotation marks as they are critical to generate json-compatible content but their meaning can change depending on the shell used. The next examples will work fine with *bash* but may not work as expected with other shells and/or OS.

Also, note that any subscription done with *mosquitto\_sub* in the command line **will let both user and password exposed in the process list** if you’re using any. When you need to provide these values (this problem disappears the moment you use certificates), the use of *mosquitto\_sub* is not recommended and a better way to communicate with the broker is by using a library.

The modifier *-v* for *mosquitto\_sub* adds the destination topic at the beginning of each received

message followed by the payload itself.

Publishing to a topic using *mosquitto\_pub*: phoenix.inf.um.es

```
mosquitto_pub -h phoenix.inf.um.es -p 1883 -u username -P password -q 2 -t "/PHOENIX-UMU/device1/attrs" -m '{"temp":25.3","hum":47.2","co2":450}'
```

Subscribing to a topic using *mosquitto\_sub* (by default with clean session, which means messages sent to this topic while the client is not connected will be lost for it):

```
mosquitto_sub -h phoenix.inf.um.es -p 1883 -u username -P password -q 2 -t "/PHOENIX-UMU/light1/cmd"
```

By default, both *mosquitto\_pub* and *mosquitto\_sub* use a random *Client ID* (see the *MQTT Configuration of Wifi Smart Sockets*) every time they connect. If you want to set one manually, just add the modifier ***-i CLIENT-ID*** with the value you prefer. Keep in mind that this identifier ***must be unique in the broker***, so don't use a random value. You should better use the name of the pilot as prefix.

If you want to subscribe using a persistent session (messages sent while the client is disconnected will be stored by the server and sent to the client once it gets online), add the modifier ***-c***. This is directly related with the previous modifier (*Client ID*) because the session is identified with the *Client ID*. So using a persistent session with a random *Client ID* makes no sense. Note that ***these commands could have been sent a very long time ago*** and therefore it might be potentially dangerous to execute them so late, which means the use of persistent sessions should be avoided when commands are used for other purposes than just sending configuration.

An extended version could be used to design a sort of *Command Manager* for a pilot using the next syntax:

```
mosquitto_sub -h phoenix.inf.um.es -p 1883 -u username -P password -q 2 -v -i "UMU-CommandManager" -t "/PHOENIX-UMU/+cmd"
```

With this subscription you will receive all the commands sent to the devices of the UMU pilot and you can do with them whatever you want. The + is the one-level wildcard while # is the multi-level wildcard. So, if you want to receive the messages sent to/by all the devices of the UMU pilot you can use the next command:

```
mosquitto_sub -h phoenix.inf.um.es -p 1883 -u username -P password -q 2 -v -t "/PHOENIX-UMU/#"
```

## **REST/JSON-LD API to communicate with Orion-LD broker of PHOENIX platform**

In this section, we will explain the REST/JSON-LD API provided by the Orion NGSI-LD broker of PHOENIX platform. By default, this API does not provide any security mechanisms. For this reason, this API will be secured by the security and privacy mechanisms that are being developed in task T4.1 of the PHOENIX project.

The Orion NGSI-LD broker developed by FIWARE open-source community has been evolved to better support linked data (entity relationships), property graphs and semantics (exploiting the capabilities offered by JSON-LD). This work has been conducted under the FIWARE and ETSI ISG CIM initiatives and has been branded as NGSI-LD. The main constructs of NGSI-LD are: Entity, Property and Relationship. NGSI-LD Entities (instances) can be the subject of Properties or Relationships. In terms of data model, Properties can be seen as the combination of an attribute and its value. Relationships allow to establish associations between instances using linked data. In practice, they are conveyed by means of a special NGSI attribute with a special value (relationship's object), which happens to be a URI which points to another entity. They are similar to the ref attributes recommended by the Data Models guidelines.

Properties and Relationships can be the subject of other Properties or Relationships. Thus, in the NGSI-LD information model there are no attribute's metadata, but just "properties of properties" or "properties of relationships". It is not expected to have infinite graphs, and in practice, only one or two levels of Property or Relationship "chaining" will happen. NGSI-LD Entities are represented using JSON-LD, a JSON-based serialization format for Linked Data. The main advantage of JSON-LD is that it offers the capability of expanding JSON terms to URIs, so that vocabularies can be used to define terms unambiguously.

Below, we provide some examples of the REST/JSON-LD interface for different operations like

entity creation, registration, data query and data query response.

## **REST/JSON-LD Examples for communicating with the PHOENIX platform**

```
curl -X POST \  
  http://<<cp_host>>/ngsi-ld/v1/entities/ \  
  -H 'Accept: application/ld+json' \  
  -H 'Content-Type: application/ld+json' \  
  -d '{  
    "@context": [  
      "http://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-  
context.jsonld",  
      "https://phoenix.inf.um.es/data-models/context.jsonld"  
    ],  
    "id": "urn:ngsi-ld:Device:UMU-Pleiades-DHW-  
temperatureSensor2",  
    "type": "Device",  
    "category": {  
      "type": "Property",  
      "value": [  
        "sensor"  
      ]  
    },  
    "description": {  
      "type": "Property",  
      "value": "Temperature sensor Secondary Circuit ->  
Solar Tank"  
    },  
    "containedIn": {  
      "type": "Relationship",  
      "object": "urn:PhoenixOntology:Zone:UMU-Pleiades-  
DHWshed"  
    },  
    "location": {  
      "type": "GeoProperty",  
      "value": {  
        "type": "Point",  
        "coordinates": [  
          38.02418,  
          -1.17311,  
          630  
        ]  
      }  
    },  
    "controlledProperty": {  
      "type": "Property",  
      "value": [  
        "temperature"  
      ]  
    },  
    "dateLastValueReported": {  
      "type": "Property",  
      "value": {  
        "@type": "DateTime",  
        "@value": "2021-04-28T11:00:00Z" } } }'
```

Figure 22: NGSI-LD Entity creation

```
curl -X POST \
  http://<<db_host>>/ngsi-ld/v1/csourceRegistrations/ \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "@context": [
      "http://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-
context.jsonld",
      "https://phoenix.inf.um.es/data-models/context.jsonld"
    ],
    "id": "urn:ngsi-ld:ContextSourceRegistration:c1",
    "type": "ContextSourceRegistration",
    "information": [{
      "entities": [{
        "id": "urn:ngsi-ld:Device:UMU-Pleiades-DHW-
temperatureSensor2",
        "type": "Device"
      }],
      "properties": [
        "category",
        "description",
        "containedIn",
        "location",
        "controlledProperty",
        "dateLastValueReported"
      ]
    }],
    "endpoint": "http://<<cp_host>>",
    "expires": "2030-11-29T14:53:15Z"
  }'
```

**Figure 23: NGSI-LD Entity registration**

```
curl -X GET \
  'http://<<db_host>>/ngsi-ld/v1/csourceRegistrations?type=Device' \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/ld+json'
```

**Figure 24: NGSI-LD query**

```
[
  {
    "id": "urn:ngsi-ld:ContextSourceRegistration:c1",
    "type": "CSourceRegistration",
    "endpoint": "http://<<cp_host>>",
    "information": {
      "entities": {
        "id": "urn:ngsi-ld:Device:UMU-Pleiades-DHW-
temperatureSensor2",
        "type": "Device"
      },
      "properties": [
        "category",
        "description",
        "containedIn",
        "location",
        "controlledProperty",
        "dateLastValueReported"
      ]
    }
  },
  ...
]
```

**Figure 25: NGSI-LD query response**



## Annex V – Example of WiFi Multi-Sensors for CO<sub>2</sub>, temperature, humidity

These devices developed and manufactured by ODINS provide readings from CO<sub>2</sub>, temperature and humidity sensors. These values are sent to the MQTT/JSON IoT Agent (see the adequate section) through WiFi network.

The configuration process requires the device to be in configuration mode. To do this, it must be opened and keep the button pushed when it gets powered on (see image below).

With a laptop or a smart-phone you should see an active WiFi network named *wimex\_XXX* that has no internet connectivity. Join the network and open the next URL in a web browser:

<http://192.168.4.1>

The *Measures* section (main screen) shows the current values for the sensors.

The *Settings* section lets you change the configuration.

The *Wireless Network Name* can be set manually or the device can search for the visible ones by clicking the button that is on the right side of the *Name* textfield.

In the *Options* section, by default, the *Device ID* is configured using the *Serial number* (visible at the beginning of the screen).



The screenshot shows the 'Settings' section of the wimex web interface. It includes fields for 'Serial number', 'MQTT Server' (with 'Host', 'Port', 'User', and 'Password' sub-fields), and 'Options' (with 'Device ID' and 'Publish interval in minutes'). There are also sections for 'Wireless Network' and 'Language'. The wimex logo is at the top, and there are tabs for 'Measures' and 'Settings'.

## Example Data model

```
{
  "@context": [
    "https://smartdatamodels.org/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
  ],
  "id": "urn:ngsi-ld:Building:buildingPLEIADES",
  "address": {
    "addressLocality": "Murcia",
    "postalCode": "30100",
    "streetAddress": "Campus de Espinardo",
    "type": "PostalAddress"
  },
  "category": [
    "office"
  ],
  "GeoProperty": {
    "coordinates": [
      [
        [
          38.02418,
          -1.17311
        ],
        [
          38.02437,
          -1.17235
        ],
        [
          38.02488,
          -1.17257
        ],
        [
          38.02468,
          -1.17332
        ]
      ]
    ],
    "type": "Polygon"
  },
  "createdAt": "2021-05-03T10:18:16Z",
  "description": "Centro de investigación",
  "floorsAboveGround": 5,
  "floorsBelowGround": 0,
  "location": {
    "GeoProperty": [
      [
        [
          38.02418,
          -1.17311
        ]
      ]
    ]
  }
}
```

```
]
  ],
  "type": "Point",
  "openingHours": [
    "Mo-Fr 10:00-19:00",
    "Sa 10:00-22:00",
    "Su 10:00-21:00"
  ],
  "type": "Building"
}
{
  "@context": [
    "https://PHOENIX-ONTOLOGY-URL/context.jsonld",
    "https://smartdatamodels.org/context.jsonld",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
  ],
  "id": "urn:PHOENIX-ONTOLOGY:Zone:Room1.14",
  "containedIn": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:Building:buildingPLEIADES"
  },
  "category": {
    "type": "Property",
    "value": "office"
  },
  "glazingArea": {
    "type": "Property",
    "value": 10
  },
  "type": "Zone"
}
{
  "id": "urn:ngsi-ld:Device:UMU-XX",
  "type": "Device",
  "category": {
    "type": "Property",
    "value": [
      "sensor"
    ]
  },
  "name": {
    "type": "Property",
    "value": "PT100/2"
  },
}
```

```

"description": {
  "type": "Property",
  "value": "Temperature sensor Room 1.14"
},
"serialNumber": {
  "type": "Property",
  "value": "XXXXX"
},
"controlledProperty": {
  "type": "Property",
  "value": [
    "temperature"
  ]
},
"refDevice": {
  "type": "Relationship",
  "object": "urn:ngsi-Id:Device:UMU-XXXX"
},
"location": {
  "type": "Relationship",
  "object": "urn:PHOENIX ONTOLOGY
URL:Zone:Room1.14"
},
"dateLastValueReported": {
  "type": "Property",
  "value": {
    "@type": "DateTime",
    "@value": "2021-04-28T11:00:00Z"
  }
},
"value": {
  "type": "Property",
  "value": "v%3D61"
},
"@context": [
  "https://smartdatamodels.org/context.jsonld"
]
}
{
  "id": "urn:ngsi-Id:lotStream:UMU-XX",
  "type": "http://purl.org/iot/ontology/iot-
stream#lotStream",
  "http://purl.org/iot/ontology/iot-
stream#generatedBy": {
    "type": "Relationship",

```

```

"object": "urn:ngsi-Id:Device:UMU-XX"
},
"https://w3id.org/iot/qoi#hasQuality": {
  "type": "Relationship",
  "object": "urn:ngsi-Id:Stream:UMU-XX"
},
"@context": [
  "https://uri.etsi.org/ngsi-Id/v1/ngsi-Id-
core-context.jsonld"
]
},
{
  "id": "urn:ngsi-Id:Observation:UMU-XX",
  "type": "http://purl.org/iot/ontology/iot-
stream#StreamObservation",
  "http://purl.org/iot/ontology/iot-stream#belongsTo": {
    "type": "Relationship",
    "object": "urn:ngsi-Id:Stream:UMU-XX"
  },
  "http://www.fault-
detection.de/hasEstimatedResult": {
    "type": "Property",
    "value": 24,
    "observedAt": "2021-02-04T08:38:43Z"
  },
  "http://www.fault-detection.de/hasVerdict": {
    "type": "Property",
    "value": "faulty"
  },
  "http://www.w3.org/ns/sosa/hasSimpleResult": {
    "type": "Property",
    "value": "18"
  },
  "http://www.w3.org/ns/sosa/observedProperty": {
    "type": "Relationship",
    "object": "urn:ngsi-
Id:controlledProperty:temperature"
  },
  "@context": [
    "https://uri.etsi.org/ngsi-Id/v1/ngsi-Id-
core-context.jsonld"
  ]
}

```