eTEACHER

D2.5: BACS add-on services container and Universal BACS Communication Interface

WP2: Empower tools I – BACS add-on services

T2.5: BACS integration through universal BACS Communication Interface

Date of document

05. November, 2019 (M25)

Authors: Florian Frank (ACX)
Gloria Calleja-Rodríguez (CEM)
eTEACHER

EE-07-2016-2017
Innovation Action

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 768738.

Disclaimer

The information reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Technical References

Project Acronym	**eTEACHER**
Project Title	end-user Tools to Empower and raise Awareness of Behavioural Change towards EneRgy efficiency
Project Coordinator	Noemi Jimenez Redondo CEMOSA noemi.jimenez@cemosa.es
Project Duration	1 October 2017 – 30 September 2020

Deliverable No.	D2.5
Dissemination Level	PU[1]
Work Package	WP 2 - Empower tools I – BACS add-on services
Task	T2.5: BACS integration through universal BACS Communication Interface
Lead beneficiary	Partner 6 (ACX)
Contributing beneficiary(ies)	
Due date of deliverable	30 September 2019
Actual submission date	30 September 2019

[1] PU = Public

PP = Restricted to other programme participants (including the Commission Services)

RE = Restricted to a group specified by the consortium (including the Commission Services)

CO = Confidential, only for members of the consortium (including the Commission Services)

Versions

Version	Person	Partner	Date
00	Florian Frank	ACX	03 September 2019
01	Florian Frank	ACX	20 September 2019
02	Florian Frank	ACX	15 October 2019
03	Florian Frank	ACX	1 November 2019
04	Florian Frank	ACX	5 November 2019

Table of Content

List of Figures

Abbreviation and Acronyms

API	Application Programming Interface
BACS	Building Automation and Control System
BEMS	Building Energy Management System
ER-Model	Entity-Relationship-Model
ICT	Information and Communication Technologies
IEQ	Indoor Environment Quality
IP	Internet Protocol
JSON	JavaScript Object Notation
OPC UA	Open Platform Communications Unified Architecture
RDBMS	Relational Database Management System
REST	Representational State Transfer
SoA	Service-oriented architecture
WiA	What-if Analysis
WP	Work Package

0 Abstract

The eTEACHER project aims to empower building occupants to achieve better energy efficiency and comfort levels and therefore enable behavioural change. To enable the users, an ICT solution is utilised, the called BACS add-ons. These BACS add-ons are designed to work with existing BACS through a universal communication interface. This deliverable describes the integration of BACS add-ons through a universal communication interface and a common database that is used to store and manage the collected building data.

The focus of the development was the common database. As all the measured data of a BACS is collected at this central point, it forms the basis of the eTEACHER solution. The BACS add-ons draw their data from this common database. The database has to be scalable to store the collected building data. To maintain a wide data pool it is crucial for the database to be interoperable with as many BACS as possible.

This is reflected in the task of creating a universal BACS Communication interface. In order to achieve this interoperability, several approaches might be feasible. Each approach is presented with specific challenges. After exploring each approach, the development of BACS add-ons was conducted.

1 Common eTEACHER database

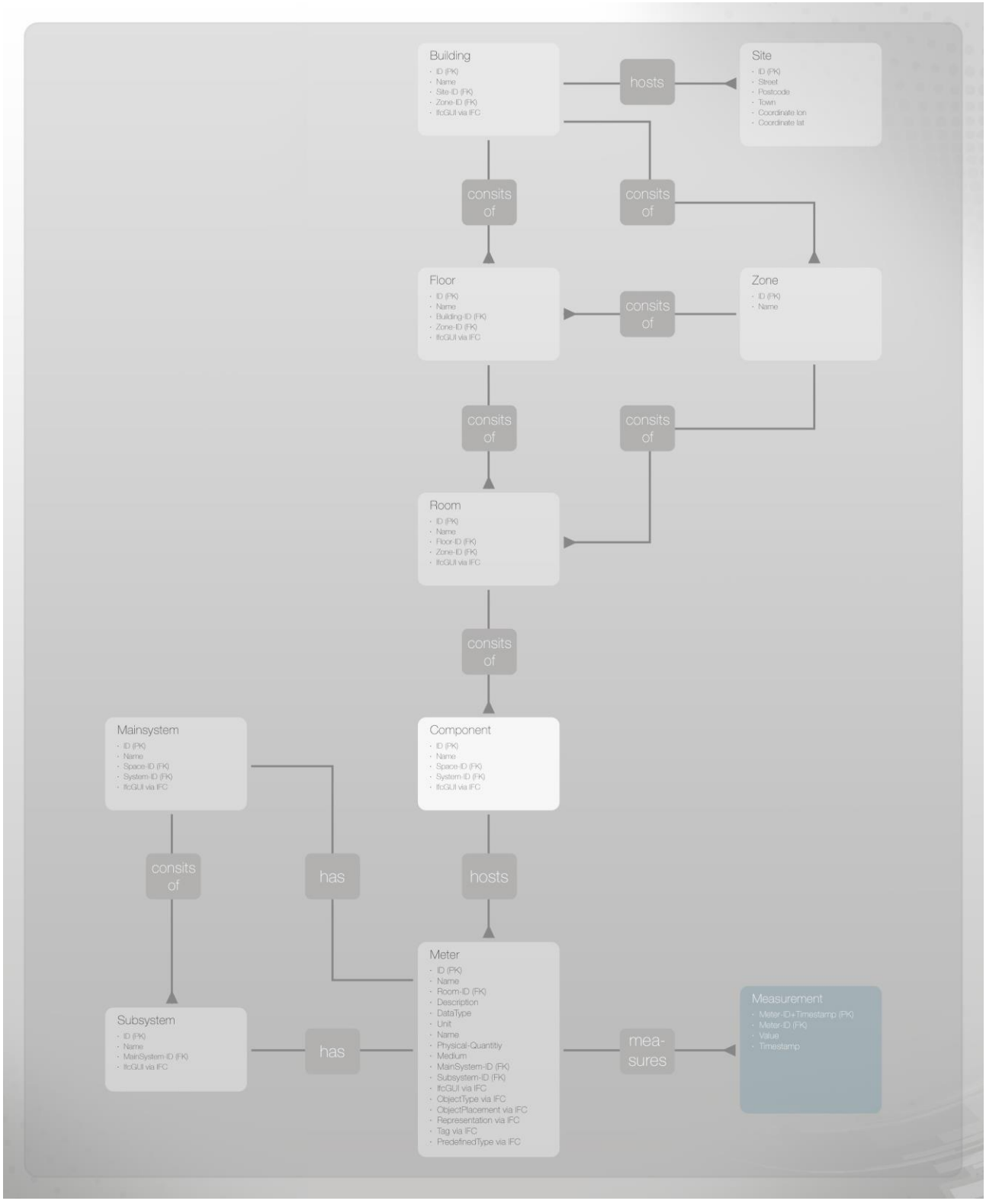
The focus of the ICT solutions in the eTEACHER project is to enable building users to change their energy behaviour. Common BACS however, consist of static rulesets, which change actuators based on sensor data. This approach does not work for eTEACHER as it does not account for the building users. Techniques like motion sensors switching lights on or off seem to be more user focused, however these solutions usually just replace common light switches, therefore limiting the users' freedom to turn lights on or off as they please.

However, there is a common ground of these conventional BACS with the eTEACHER approach - sensor data. The sensor data will not be used to directly influence actuators, though. Sensor data is collected to serve as a base for the eTEACHER add-ons.

The following paragraphs aim to explain the steps taken when creating the common database and provide a basic understanding of databases and the underlying concepts. Explanations are focused on clarifying the structure and the decisions behind the database concepts for the use in the eTEACHER project.

1.1 Entity relationship model

The eTEACHER common database is a relational database. A relational database is a collection of tables that are related with each other. A table is a collection of structured data that is listed in columns and rows. Each intersection of a column with a row is called a cell. The columns have properties which defines what kind of data, if any, can be stored within. The ER-model describes the relations between the entities of the database.



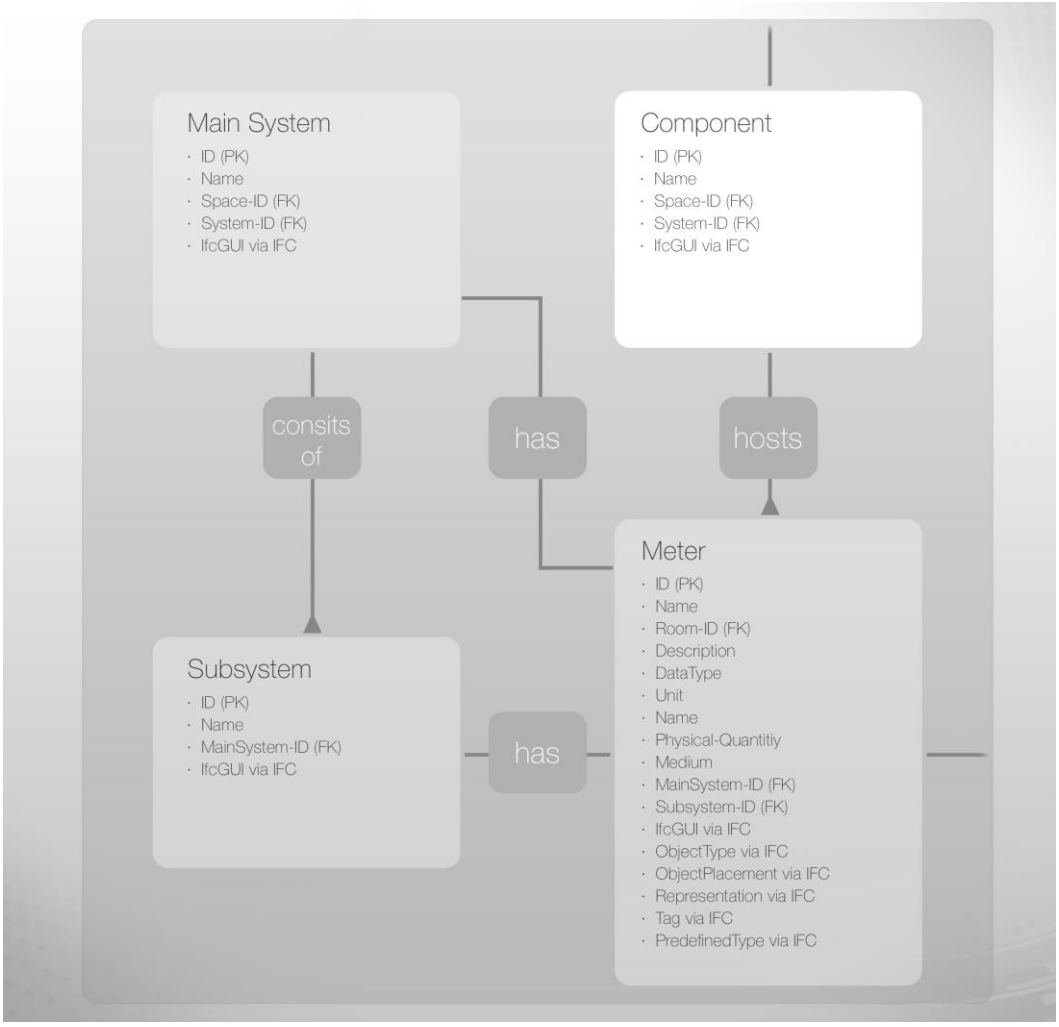
1 ER-Modell of eTEACHER common database

1.1.1 Entities

The entity relationship model is a methodology used to create relational databases. This is applicable for the eTEACHER common database as is supposed to collect the data of real-life entities, such as sensors or buildings. Following this approach, before creating the actual database, all the applicable entities were identified and collected. Some examples include:

- Building locations
- Rooms inside buildings
- Meters

Collecting the entities was done in close communication with the project technical partners, since missing any necessary entity would highly impact their add-ons. During this task, the usage of the Building Information Modelling (BIM) was discussed. As BIM is a software assisted process based on 3D-modelling of a building it was discarded because this would limit the flexibility of eTEACHER. Besides, if the target building did not have and existing 3D-model, it would have to be created to deploy the solution, increasing the installation time and the operation costs for each building. This would have been the case with all of the eTEACHER's pilot buildings.



2 Attributes of the 'Meter' table of eTEACHER's common database

1.1.2 Attributes

After the entities were listed, their attributes (properties) were specified. This process defines the appearance of the tables within the database as each attribute is represented as a column. Each attribute has to be applicable for each row of the table. Using the example of the table “sites” the attributes are

- Name
- Street (address)
- Postcode (address)
- Town (address)
- Country (address)

Other important aspect when creating attributes of the tables of the common database for eTEACHER was the requirement of scalability. The aforementioned BIM process that was originally discarded had an influence on the properties of certain tables. The IFC-Standard, developed by the buildSMART [2]international foundation, specifies identifiers for each element in ifc-files. The ‘”room” or “meters” table for example, are designed with the attributes/column “ifcGUI” which allows to store the corresponding identifier of a ifc-file.

1.1.3 Relationships

The relationships are the logical connection between the tables (entities). Additional to the attributes that represent the properties of the real-life entities (refer to 2.1.2 Attributes) there are keys in each table. Just as attributes, keys are listed as a column of the table. Each table bears a key that is unique for each row of the table. This key is called the primary key. In the eTEACHER database the primary key for each table is stored as an integer in a column “ID”.

Relationships are created by listing the primary key of one table as column in another table. It is then called foreign key. In eTEACHER for example, the table “buildings” lists the integers 1 to 12 for all twelve pilot buildings as primary key. Additionally there is the column “Site_ID” which relates the pilot building to the site it is located at and is therefore a foreign key.

1.2 Structured Query Language SQL

SQL is a standardised programming language designed for managing data stored in the relational database management systems. Whilst the eTEACHER solution does not require knowledge of SQL to work, it is mentioned here as it allows to work with the data of eTEACHER. Amongst other things SQL allows to create, read, update and delete data. Some of the add-ons are based on SQL.

1.3 Relational Database Management System

The term Relational Database Management System describes the software that is used to maintain relational databases.

When creating the common eTEACHER database, the underlying RDBMS had to be chosen. Solutions by Microsoft and ORACLE were evaluated. The consortium decided to use the RDBMS

[2] https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/

solution of ORACLE: MySQL[3]. As both versions were equally fit to support eTEACHER, the decision was influenced by the interoperability with the additional sensors used for collecting data in the pilot buildings.

MySQL is long standing and open source, therefore well established, well documented and proven and reliable. The use of open source software allows the ICT community to extend the platform's functionalities and assure the legacy of its components.

1.4 Deviations and Challenges

Deviating from the plan of a single common database, project partners established national databases which introduced specific challenges for the work on the common database. The national databases were introduced at the evaluation phase for the RDBMS and before the ER-model was finished. They are used to store the data of additional sensor installed in the pilot buildings.

The common database therefore had to be compatible with the national databases, so it could be synchronised with the national databases. Additionally, since the national databases do not strictly follow the ER-model of the common database, a synchronisation had to be manually adjusted for that.

In order to create the compatibility of the database, all the databases use the same RDBMS. For synchronisation, an extra BACS add-on was developed.

[3] https://www.mysql.com/

2 Universal BACS Communication Interface development

The range of BACS is very broad and there is not one universal overarching BACS operating system but there are several ecosystems on the market and a building can often include a multitude of them. This poses challenges for the eTEACHER project.

If the eTEACHER project was to pursue the integration with a specific ecosystem respectively vendor it would be depending on this ecosystem being used in the building in which the eTEACHER solution is to be integrated. Retrofitting such a system into a building is connected with additional financial investments. Another drawback would be the maintainability. If the chosen ecosystem was to vanish from the market, the depending solutions could not be maintained anymore.

Therefore, a universal BACS Communication Interface is a core aspect of task 2.5 in eTEACHER. The goal is to achieve interoperability with ideally all ecosystems available at the market, allowing the eTEACHER solution to be installed into any building with an already existing BACS at minimal costs.

To achieve this, the UBCI needs to be able to communicate with existing BACS and transfer data to the BACS add-ons of the eTEACHER solution, referred to as interoperability. The UBCI focuses on two levels of interoperability which were defined in an ETSI whitepaper[4] : **technical interoperability,** which is associated with hardware/software components, systems and platforms and the communication protocols these components use to communicate and **syntactical interoperability** which refers to the data formats that are transferred via the aforementioned protocols. These concepts of interoperability can be compared to the Open Systems Interconnection Model[5], where higher layers (here syntactical interoperability) are based on lower layers (here technical interoperability).

2.1 Technical interoperability

In order to create the technical interoperability an underlying protocol had to be chosen. Choosing an existing BACS protocol would mean to establish a dependency to a single ecosystems. As mentioned before, this was not a feasible option. Creating a new protocol would mean adding to the already existing multitude of ecosystems. Additionally, a new protocol had to either be adapted by the manufacturers of BACS or defined in a way that syntactical interoperability could be achieved within the project. Considering the amount of existing data formats for BACS, some of them public standard (e.g. DALI[6]), some of them available for paying members only (e.g. KNX)[7] and even some proprietary, this approach was could not be realised within the scope of the eTEACHER project.

[4] https://www.etsi.org/images/files/ETSIWhitePapers/IOP%20whitepaper%20Edition%203%20final.pdf

[5] https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=2820

[6] https://www.digitalilluminationinterface.org/

[7] https://www.knx.org/

A compromise was found by choosing an already existing protocol that does not directly prefer one of the already existing BACS and ideally can be used for communication with the BACS add-ons without the need for the add-ons to be further adjusted: the internet protocol[8].

The internet protocol was first published in 1974 by the Institute of Electrical and Electronics Engineers (IEEE)[9], the 4th version (IPv4), which was described by the IETF[10] in 1981 is one of the core protocols that make up the internet and therefore well established. It is an open standard (RFC 791)[11] that is implemented by many recently developed smart home devices/products that make up the internet of things[12]. The only drawback for the eTEACHER project that was identified is that existing BACS installations might not currently support the internet protocol. This can be solved by installing gateway-devices when eTEACHER is implemented into buildings with existing BACS. In fact, the eTEACHER project makes use of gateways to log data into the common database.

2.2 Syntactical interoperability

After technical interoperability was achieved, syntactical interoperability had to be implemented. Many existing BACS standards have specified data formats for usage with IP such as (KNX[13] or Modbus[14]) which would have to be implemented. The software that implements these data formats for the eTEACHER project is the BACS “ViciOne” by technical partner ACX GmbH (refer to 3.1).

Not all data formats are publically available, however. Some are proprietary and can only be accessed by paying license fees to the owner, if they are available at all. To still achieve syntactical interoperability, other data formats have to be used as substitute. To do so, one could either install more gateways or create a new data format. Whilst both approaches are feasible, within the eTEACHER project the development for a new data format was preferred. On the one hand, installing more gateways would be more expensive and would add more sources of defect, on the other hand, the BACS add-ons on would need a data format to communicate with the common database. The new data format was specified as the BACS add-on API (refer to 3.2).

[8] https://tools.ietf.org/html/rfc791

[9] https://www.ieee.org/

[10] https://www.ietf.org/

[11] https://www.rfc-editor.org/info/rfc791

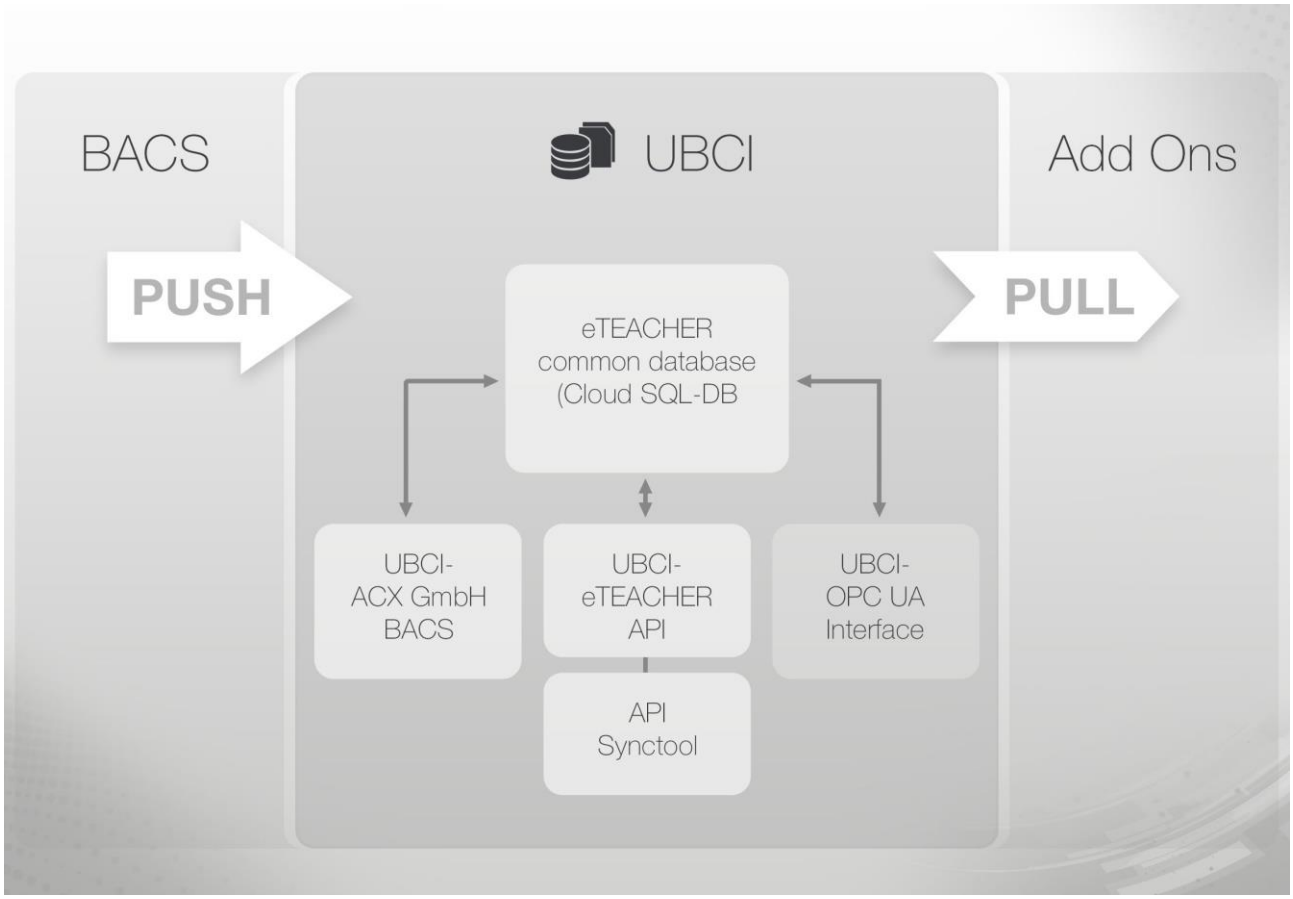
[12] https://ec.europa.eu/digital-single-market/en/internet-of-things

[13] https://www.knx.org/

[14] http://www.modbus.org/

3 Universal BACS Communication Interface add-ons

The eTEACHER project utilizes different software approaches to achieve universal interoperability. Each approach is covered by a communication BACS add-on: existing BACS data formats are covered by ACX GmbH's ViciOne, the eTEACHER projects, data formats are covered by both, the eTEACHER API or the OPC UA interface add-on. Existing BACS can push their sensor data to the common database using these add-ons. The eTEACHER BACS add-ons can use the communication add-ons to pull data needed from the eTEACHER common database.



3 Data flow of the UBCI Container

3.1 ACX GmbH BACS ViciOne

The product ViciOne is the BACS of partner ACX GmbH. It consists of hardware and software solutions. As illustrated before, ViciOne plays a key role of creating syntactical interoperability with existing BACS. The data formats for communication with KNX[15], EnOcean[16] and BACnet[17] devices as well as devices using the MQTT[18] protocol via IP-Interfaces are implemented.

BACS hardware that is already installed in the building is abstracted to a software functionblock of the ViciOne Software Automation Suite. This can be done manually, utilising a library of functionblocks as well as using a software-wizard that creates functionblocks from existing project databases, the likes of which are used in the KNX Standard.

This allows the software to read sensor data. Further functionblocks can be used to process the data, e.g. converting the unit of measured values. Finally a set of functionblocks enhancing the software to support the eTEACHER common database using SQL (refer to 1.2) logs the read sensor data into the common eTEACHER database.

3.2 eTEACHER API

The eTEACHER application programming interface (API) is used to query as well as update data of the common database. An API is a common and modern technique to provide interfaces for exchanging data in the so called internet of things (IoT). The basic function is similar to how the World Wide Web works: A user sends a request for information to the API, which then queries the common database and presents the results of the query to the user. This is also referred to as Representational State Transfer, or REST[19].

The API can be used via web browsers as well as other software, which makes it vendor independent and universal. It was developed to meet the projects requirements for syntactical interoperability and is a fully-fledged communication add-on. It was used as a backend for an eTEACHER project specific website that allowed the pilot coordinators to update the pilot buildings' data in the common database as a proof of concept. Another use is the communication with the BACS add-ons of the other technical partners. Instead of developing an implementation for SQL for each individual BACS add-on the eTEACHER API centralises this implementation. This would allow for changing the RDBMS (refer to 1.3) of the common database with minor adjustments: instead of adjusting all the partner's BACS add-ons, only the eTEACHER API would have to be adjusted.

[15] https://www.knx.org/

[16] https://www.enocean-alliance.org/

[17] http://www.bacnet.org/

[18] http://www.mqtt.org

[19] https://searchapparchitecture.techtarget.com/definition/REST-REpresentational-State-Transfer

The API is based on ASP.NET Core[20] programmed in C# and implements the MySQL[21] class maintained by Oracle Corporation. It is currently utilised by the eTEACHER BACS add-on "Data Processing for systems performance and indoor environmental quality" to read data from the common database. The BACS-solution developed for one of the eTEACHER pilots (residential building block InCity located in Bucharest, Romania) used the eTEACHER API to push the data measured locally to the common eTEACHER database.

The documentation of the API is an appendix to this deliverable.

3.2.1 eTEACHER SyncTool

The eTEACHER SyncTool is a stand-alone add-on based on the aforementioned API. Deviating from the project plan, pilot sites are maintaining local databases. It is necessary to update the eTEACHER common database with the data stored in this local databases.

The SyncTool add-on was developed to solve the synchronisation issues. Using the MySQL implementation of the eTEACHER API it queries the local databases and updates the common database with the results through the API.

Even though the add-on is currently specialised on the structure of the databases used at the pilot sites it is an invaluable add-on for the UBCI: With minimal updates to the query logic it can be used to query databases used by other BACS and convert the data for the common database.

The SyncTool add-on is used in several instances, querying tables in both the UK and Spain, updating the common database every 10 minutes.

3.3 eTEACHER OPC UA

Open Platform Communication Unified Architecture is a protocol designed for machine-to-machine communication in industrial requirement. provides a standardized, open, cross platform and service-oriented architecture (SOA), which complements the API add-on.

The OPC Server which is part of the UBCI-Container includes an integral information model to provide meta information over the available data. A client can browse this information model and read underlying information like actual and historical sensor values. The server provides memory space in its information model where a client can store any kind of data.

The OPC Server follows the architecture of the common database to provide data points. Each data set in a specific table of common database represents a data point. The OPC Client, implemented in the What-if-Analysis BACS add-on subscribes to this data points and makes them available for the add-on. The implementation of the OPC Server would allow to add more data points using an OPC Client, thus allowing third parties to achieve syntactical interoperability with the eTEACHER project.

[20] https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core

[21] https://www.nuget.org/packages/MySql.Data/

4 BACS add-on services container

The BACS add-on services container is the concept of all necessary software programs being concealed in a single environment. The idea for such a services container is that eTEACHER is easy to roll out and maintain. Instead of users, administrators or system integrators having to install several tools and then adjusting them, they ideally should not need to set up anything but the BACS add-on services container itself, thus saving time and funds.

The need for creating the BACS add-on services container arises due to the ICT nature of the eTEACHER project. Rather than implementing a single piece of hardware, the BACS add-ons are software solutions. They could be packaged with a server hardware, however this would add more costs when installing the eTEACHER solutions in building that already have an ICT environment attached.

A common concept of dissolving the hardware dependency of software packages would be virtualisation[22]. Virtual machines are a software image that abstracts a physical computer. Utilising a Hypervisor software, a server computer can then run several virtual machines. A virtual machine image includes a full copy of the operation system as well as all applications installed, necessary drivers, libraries and binaries. This technique is used in data centres, where a customer can rent a server machine to use. Instead of having huge quantities of small server computers, a smaller number of server computers with huge quantities in processing power, Random Access Memory and storage space are used as hypervisors that run the virtual machines. This effect reduces hardware costs and scales down even to small businesses.

Even though the technique of virtualisation was feasible for the eTEACHER project was used. During the course of the project it was discussed, that the eTEACHER solution could be more flexible if not all BACS add-ons would needed to be used. This allows the solution to be tailored more to the needs of the users of the solution. This desirable flexibility deviates from the original project plan and would complicate the use of virtual machines, as a multitude of images with different combinations of the add-ons would have to be created and maintained.

In 2013 the container software Docker[23] was released by Docker, Inc. In contrast to the aforementioned virtual machines, software containers do not abstract a physical computer, but are abstracting at the app layer. Code and its dependencies are packaged as a software image. Instead of a hypervisor, they utilise the container software Docker to run, which is installed on the host operation system. Since the container image included all the dependencies, it is independent from the OS, therefore portable and scalable, can be developed and maintained on any operation system.

Even though instead of a single service container (virtual machine) there are now several software containers, the flexibility and ease of maintenance of this software containers lead to the decision that within the scope of eTEACHER, the add-ons created in Task 2.5 use the software containerisation technique of Docker.

[22] https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-virtual-machines.html

[23] https://www.docker.com/

5 Conclusion

This report describes the results of T2.5 of the eTEACHER project for designing and implementing the BACS add-on services container and Universal BACS Communication Interface of eTEACHER, as well as the underlying common database. The process of creating the common database was described. Even though relational databases are basic principle of electronic data processing, sculpting a database for the use of different add-ons within the eTEACHER project has been a challenge. Through the development of the project, the ER-model has proven itself scalable and flexible to be updated to the changing requirements of the connected add-ons.

Seamlessly integrated with the common databases are the add-ons of the UBCI container, making the database the core of it. All add-ons are providing a way to create, read and update the data, whilst offering different interfaces to integrate other BACS or be integrated into BACS themselves. Using the software container technology when creating the UBCI add-ons makes them versatile and scalable. This is proven by the eTEACHER project actually utilising all of the add-ons in different technical contexts.

The UBCI can be used in parts or whole in different projects, both utilising all, a few, or none of the other add-ons which makes it exploitable beyond the eTEACHER projects. Further development will be aimed towards an energy reporting tool, utilising the ER-model of the common database combined with the ACX GmbH BACS add-on and a newly developed user interface.

6 References

Docker Inc. (2019) Why Docker? Retrieved from https://www.docker.com/why-docker

Donald Firesmith, Carnegie Mellon University (September 18,2017) Virtualization via Virtual Machines Retrieved from https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-virtual-machines.html

Digital Illumination Interface Alliance (2019) Introducing Dali Retrieved from https://www.digitalilluminationinterface.org/dali/

European Commission (2019) The Internet of Things Retrieved from https://ec.europa.eu/digital-single-market/en/internet-of-things

European Telecommunications Standards Institute (April 2008) Etsi White Paper No.3 Achieving Technical Interoperability – the ETSI Approach Retrieved from https://www.etsi.org/images/files/ETSIWhitePapers/IOP%20whitepaper%20Edition%203%20final.pdf

Information Sciences Institute University of Southern California (1981) INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION Retrieved from https://www.rfc-editor.org/rfc/pdfrfc/rfc791.txt.pdf

International Telecommunication Union (ITU) (1994) ITU-T X.200 (07/1994) Information technology – Open Systems Interconnection – Basic Reference Model: The basic model Retrieved from https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=2820

KNX Association cvba (2019) A brief introduction to KNX Retrieved from https://www.knx.org/knx-en/for-professionals/What-is-KNX/A-brief-introduction/index.php

Modbus Organization, Inc (2019) Modbus TCP Toolkit Retrieved from http://www.modbus.org/toolkit.php

OPC Foundation (2019). Unified Architecture. (n.d.). Retrieved from https://opcfoundation.org/about/opc-technologies/opc-ua/

Oracle Corporation (2019) Why MySql? Retrieved from https://www.mysql.com/why-mysql/

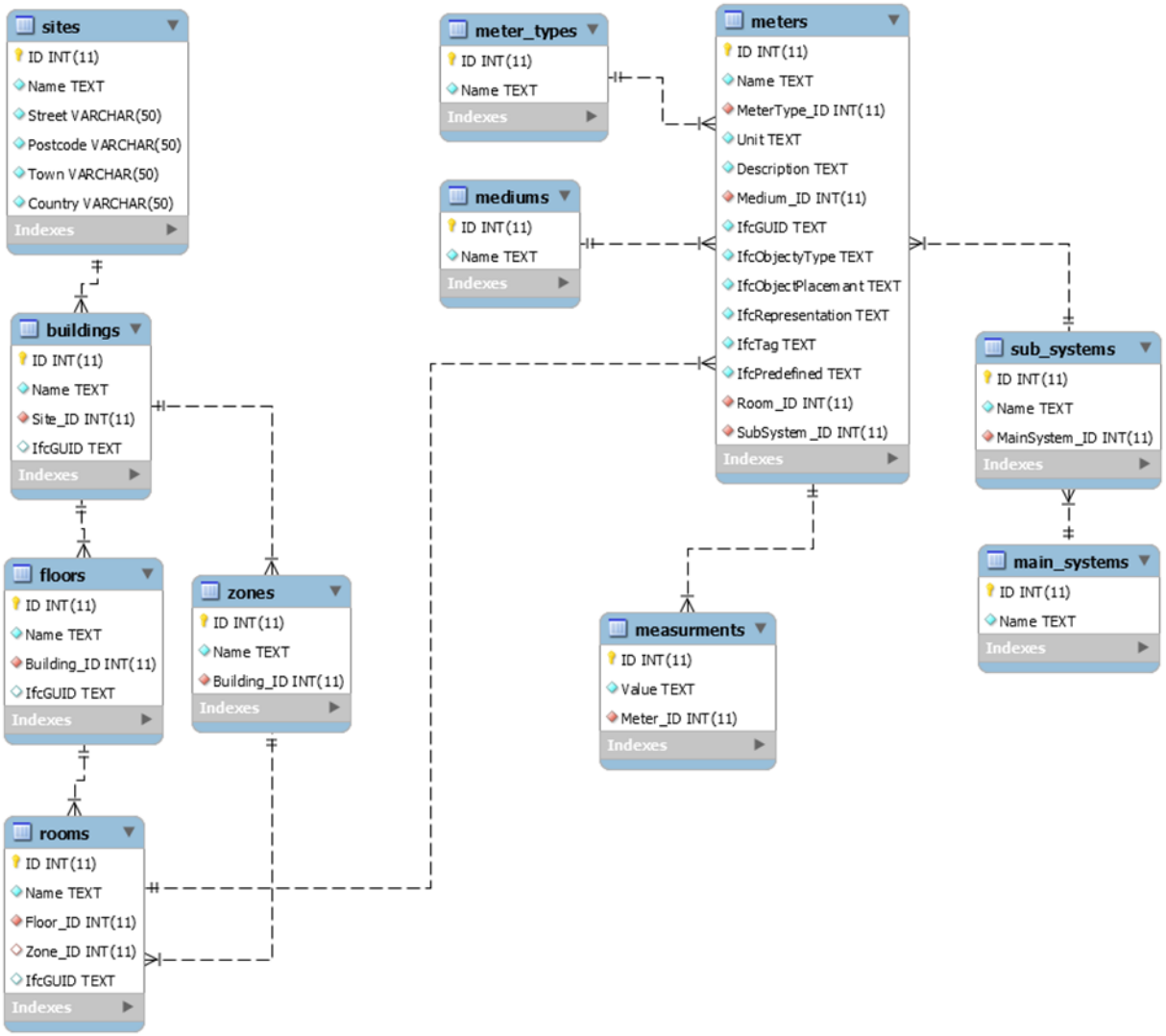
Annex A1. eTEACHER REST API documentation

1 Introduction

In this documentation is a brief description about the eTEACHER common database and its REST API. The REST-API is based on the tables of the common database and follows the syntactical suggestions for the eTEACHER BACS add-on ‘Data Processing for systems performance and indoor environmental quality’.

2 Database

2.1 ER-Model



2.2 Listing

Table	Column	Description	Data type	Granlund API name
sites	ID (PK)	-	INT(11)	facilityId
	Name	-	TEXT	facilityName
	Street	max length 50	VARCHAR(50)	-
	Postcode	max length 50	VARCHAR(50)	-
	Town	max length 50	VARCHAR(50)	-
	Country	max length 50	VARCHAR(50)	-
buildings	ID (PK)	-	INT(11)	BuildingId
	Name	-	TEXT	BuildingName
	Site_ID (FK)	References sites	INT(11)	-
	IfcGUID	-	TEXT	-
floors	ID (PK)	-	INT(11)	floorId
	Name	-	TEXT	StoreyName
	Building_ID (FK)	References buildings	INT(11)	-
	IfcGUID	-	TEXT	-
rooms	ID (PK)	-	INT(11)	roomId
	Name	-	TEXT	RoomName
	Floor_ID (FK)	References floors	INT(11)	-
	Zone_ID (FK)	References zones	INT(11)	-
	IfcGUID	-	TEXT	-
meters	ID (PK)	-	INT(11)	Device-ID
	Name	-	TEXT	DeviceName
	Meter_Type (FK)	References meter_types	INT(11)	-
	Unit	-	TEXT	-
	Description	-	TEXT	-
	Medium_ID (FK)	References mediums	INT(11)	Physical-Quantity
	IfcGUID	-	TEXT	-
	IfcObjectType	-	TEXT	-
	IfcRpresentation	-	TEXT	-
	IfcTag	-	TEXT	-
	IfcPredefined	-	TEXT	-
	Room_ID (FK)	References rooms	INT(11)	-
	SubSystem_ID (FK)	References sub_systems	INT(11)	-

Table	Column	Description	Data type	Granlund API name
measurements	ID (PK)	-	INT(11)	-
	Value	-	TEXT	-
	Timestamp	-	TIMESTAMP	
	Meter_ID (FK)	References meters	INT(11)	-
zones	ID (PK)	-	INT(11)	-
	Name	-	TEXT	-
	Building_ID (FK)	References buildings	INT(11)	-
sub_systems	ID (PK)	-	INT(11)	Subsystem-ID
	Name	-	TEXT	SubsystemName
	MainSystem_ID (FK)	References main_systems	INT(11)	-
main_systems	ID (PK)	-	INT(11)	MainSystem-ID
	Name	-	TEXT	MainSystemName
mediums	ID (PK)	-	INT(11)	-
	Name	-	TEXT	-
meter_types	ID (PK)	-	INT(11)	-
	Name	-	TEXT	-

3 REST-API

The API requires a Basic Authentication. The Username and Password are equal to database access and the API uses this for database requests.

3.1 REST-API Endpoints

The API supports default REST-API Methods GET, POST, PUT and DELETE for every endpoint.

3.1.1 List of Endpoints

Controller	Method	Address
sites	GET	/api/v1/sites
	GET	/api/v1/sites/{id}
	POST	/api/v1/sites
	PUT	/api/v1/sites
buildings	GET	/api/v1/buildings
	GET	/api/v1/buildings/{id}
	POST	/api/v1/buildings
	PUT	/api/v1/buildings
floors	GET	/api/v1/floors
	GET	/api/v1/floors/{id}
	POST	/api/v1/floors
	PUT	/api/v1/floors
rooms	GET	/api/v1/rooms
	GET	/api/v1/rooms/{id}
	POST	/api/v1/rooms
	PUT	/api/v1/rooms
meters	GET	/api/v1/meters
	GET	/api/v1/meters/{id}
	POST	/api/v1/meters
	PUT	/api/v1/meters
measurements	GET	/api/v1/measurements
	GET	/api/v1/measurements/{id}
	POST	/api/v1/measurements
	PUT	/api/v1/measurements
zones	GET	/api/v1/zones
	GET	/api/v1/zones/{id}
	POST	/api/v1/zones
	PUT	/api/v1/zones
subsystems	GET	/api/v1/subsystems
	GET	/api/v1/subsystems/{id}
	POST	/api/v1/subsystems
	PUT	/api/v1/subsystems
mainsystems	GET	/api/v1/mainsystems
	GET	/api/v1/mainsystems/{id}
	POST	/api/v1/mainsystems
	PUT	/api/v1/mainsystems
mediums	GET	/api/v1/mediums
	GET	/api/v1/mediums/{id}
	POST	/api/v1/mediums
	PUT	/api/v1/mediums
metertypes	GET	/api/v1/metertypes
	GET	/api/v1/metertypes/{id}
	POST	/api/v1/metertypes
	PUT	/api/v1/metertypes

3.1.2 Example Objects

Example JSON response for GET:


```
[
	{
		"id": 1,
		"name": "name",
		"street": "street",
		"postCode": "postcode",
		"town": "town",
		"country": "country"
	}
]
```


Example JSON response for GET by ID:


```
{
	"id": "id",
	"name": "name",
	"street": "street",
	"postCode": "postcode",
	"town": "town",
	"country": "country"
}
```

Example JSON for POST:


```
{
	"name": "name",
	"street": "street",
	"postCode": "postcode",
	"town": "town",
	"country": "country"
}
```


Example JSON for PUT:


```
{
	"id": "id",
	"name": "name",
	"street": "street",
	"postCode": "postcode",
	"town": "town",
	"country": "country"
}
```


3.2 Special Endpoints

3.2.1 Rooms (≈ GetBuildingsRooms)

Method: GET

Address: /api/v1/rooms/filter

Parameters

Name	Optional	Description	Data Type
buildingId	Yes	Buildings identifier	INT 32
floorid	Yes	Floor identifier	INT 32
medium	Yes	Medium	INT 32

3.2.2 Meters (≈ GetFacilitiesMeasurePoints)

Method: GET

Address: /api/v1/meters/filter

Parameters

Name	Optional	Description	Data Type
siteId	Yes	Site identifier	INT 32
meterId	Yes	Meter identifier	INT 32
getLatestData	Yes	Return additionally to meter object the latest Value	BOOL

3.2.2 Measurements (≈ GetBuildingsRoomsConditions, GetBuildingsConditions)

Method: GET

Address: /api/v1/measurements/filter

Parameters

Name	Optional	Description	Data Type
buildingId	Yes	Buildings identifier	INT 32
meterId	Yes	Floor identifier	INT 32
from	Yes	Returns only greater values or this value	DATETIME
to	Yes	Return only smaller values or this value	DATETIME
period	Yes (default value)	Defines how values are summarized and default value is d. min (minutes), quarter-hour, h (hours), d (days), week, a (years)	STRING
mediumId	Yes	Mediums identifier	INT 32
roomIds	Yes	Rooms identifiers	Array of INT 32

3.2.3 Post list of measurements

Method: POST

Address: /api/v1/measurements/list

This endpoint can be used to add a list of measurements.