Sensor integration tutorial

Citisim

Smart City 3D Simulation and Monitoring Platform

Sensor integration tutorial

ITEA3 – Project Citisim

Document Properties

Edited by:	Félix Jesús Villanueva Molina (UCLM)
Authors	Citisim Partners
Date	09/12/2019
Visiblity	Public
Status	Final version

History of Changes

Release	Date	Author, Organization	Changes
1.0	9/12/2019	Abalia	Document

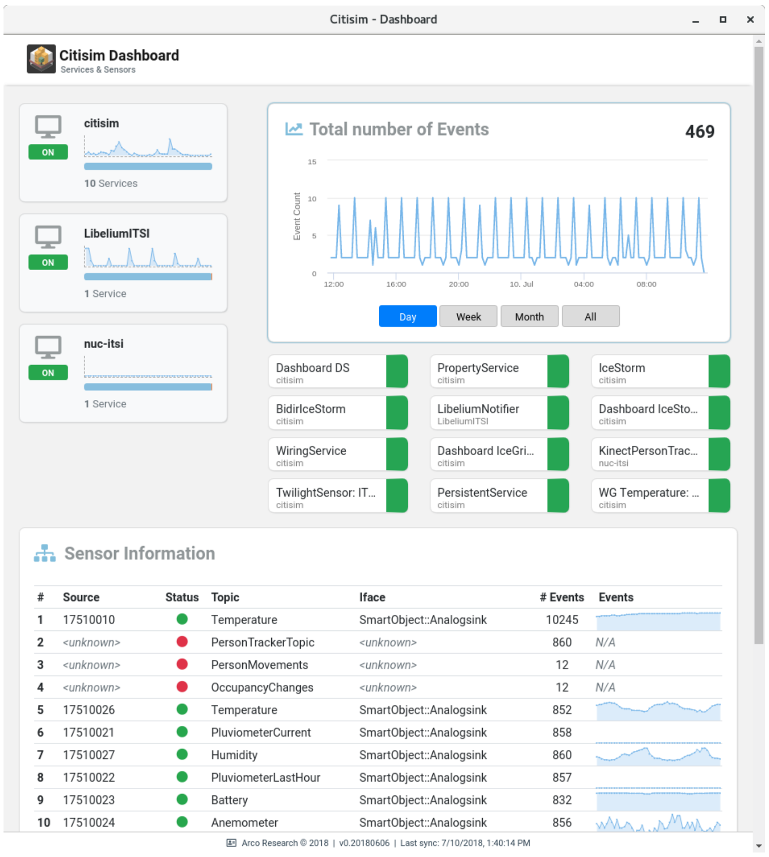
List of Figures

Table of Contents

Sensor integration tutorial

Introduction

This document presents how to integrate any sensor in Citisim, as example, we show the integration of Wibeee sensors



Dashboard DS (citisim)	🟩
PropertyService (citisim)	🟩
IceStorm (citisim)	🟩
BidirIceStorm (citisim)	🟩
LibeliumNotifier (LibeliumITSI)	🟩
Dashboard IceSto… (citisim)	🟩
WiringService (citisim)	🟩
Dashboard IceGri… (citisim)	🟩
KinectPersonTrac… (nuc-itsi)	🟩
TwilightSensor: IT… (citisim)	🟩
PersistentService (citisim)	🟩
WG Temperature: … (citisim)	🟩

Sensor Information

#	Source	Status	Topic	Iface	# Events	Events
1	17510010	●	Temperature	SmartObject::Analogsink	10245	
2	*<unknown>*	●	PersonTrackerTopic	*<unknown>*	860	*N/A*
3	*<unknown>*	●	PersonMovements	*<unknown>*	12	*N/A*
4	*<unknown>*	●	OccupancyChanges	*<unknown>*	12	*N/A*
5	17510026	●	Temperature	SmartObject::Analogsink	852	
6	17510021	●	PluviometerCurrent	SmartObject::Analogsink	858	
7	17510027	●	Humidity	SmartObject::Analogsink	860	
8	17510022	●	PluviometerLastHour	SmartObject::Analogsink	857	
9	17510023	●	Battery	SmartObject::Analogsink	832	
10	17510024	●	Anemometer	SmartObject::Analogsink	856	

Arco Research © 2018 | v0.20180606 | Last sync: 7/10/2018, 1:40:14 PM

IoT layer in Citisim has been developed with the current business IoT models in mind, so we can find three cases when we need to meter/control a physical magnitude in a Citisim instance:

1. We buy a programable sensor/actuator. In this case we can native integrate Citisim API calls from the sensor directly.

2. We buy a non-programable sensor/actuator which it is using a standard/open protocol (e.g. MQTT). In this case we need to run an adapter (e.g. MQTT-Citisim adapter) and to configure it properly.

3. We buy a non-programable sensor/actuator which sends all the information to a fixed, non-configurable, cloud service. We are going to deal in this lab with this case.

Example of Wibeee sensor integration.

Effectively, in last years it is a usual practice in a lot of hardware companies to sell you devices and then pay monthly for accessing to the data of your devices. For the final user and it depends on the final application, this approach would be a good approach due they don't have to be worried about dashboard issues/maintenance. However, for ingesting data platforms as Citisim platform is, this is the worst scenario.

Under this approach and from a technical point of view, there are only two options:

- Some companies provide you with an API (usually an HTTP REST API) to access to the data, directly from the sensor and/or against the cloud.

- To make web scraping of the dashboard, we highly recommend you to avoid this type of hardware in case you are buying new sensors/actuators. If you are dealing with legacy systems be aware about legal issues to ingest data using web scrapping.

In the first approach where you have an HTTP REST API to access the data, the best option is to develop something similar to an adapter for Citisim from that technology.

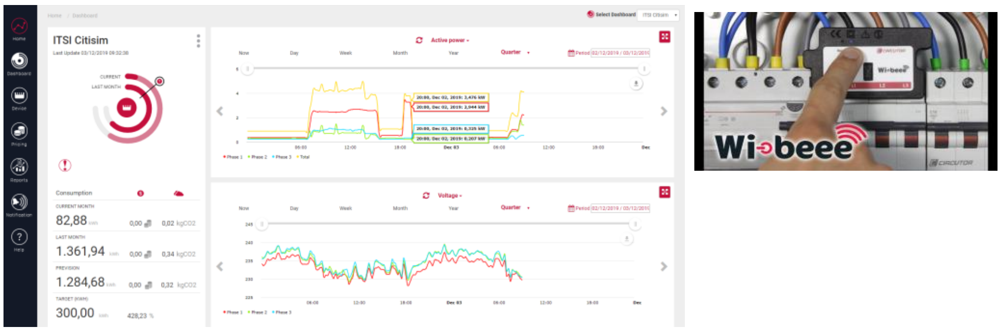


Figure 1 Wibee Sensor and its dashboard in Wibeee cloud

Let’s see step by step how to deal with this scenario using wibeee company energy sensors. The company Wibeee has catalog of high quality energy sensors to analyze energy consumption from a smart mobile app.

The steps you have to follow are:

1. To buy the sensors from its web.
2. To deploy them in the electrical distribution panel,
3. To configure the Wibeee device to connect to your wifi
4. Finally you can log-in in its cloud service in order to see your data (http://wibeee.circutor.com/wibeee/) and/or log-in with app from the mobile phone.
5. Alternatively you can download periodically a xml file with the data of your sensor, we choose this approach and we are going to explain how to inject the information collected by this (or anyother way as HTTP REST API as we mention above) in a Citisim instance.

The first part, download periodically the xml file with the desired wibeee information can be done with any python library, we are not going in depth in this part since it is specific of each sensor. The reason why we chosen wibeee is because you can access directly to the sensor to get the information directly from the sensor so you don’t need to access to the cloud.

In a config file (e.g publisher.config) we need to stablish the parameters to be connected to a Citisim instance and to the sensors:

1. Ice.Default.Locator = IceGrid/Locator -t:tcp -h <host> -p 5061
2. LibCitisim.UseBiDir = True #if we are behind a gateway
3. Wibeee.url.citisim = http:*//X.X.X.X/services/user/values.xml?id=ITSI%20Citisim*
4. Wibeee.url.corridor = http:*//X.X.X.X/services/user/values.xml?id=ITSI%20Pasillo*
5. Source.ids.citisim = {"pact":"0A06175100000080", "papt":"0A06175100000081", "fpott":"0A06175100000082", "irmst":"0A06175100000083", "eact":"0A06175100000084", "vrmst":"0A06175100000085"}
6. Source.ids.corridor = {"pact":"0A06175100000086", "papt":"0A06175100000087", "fpott":"0A06175100000088", "irmst":"0A06175100000089", "eact":"0A06175100000090", "vrmst":"0A06175100000091"}

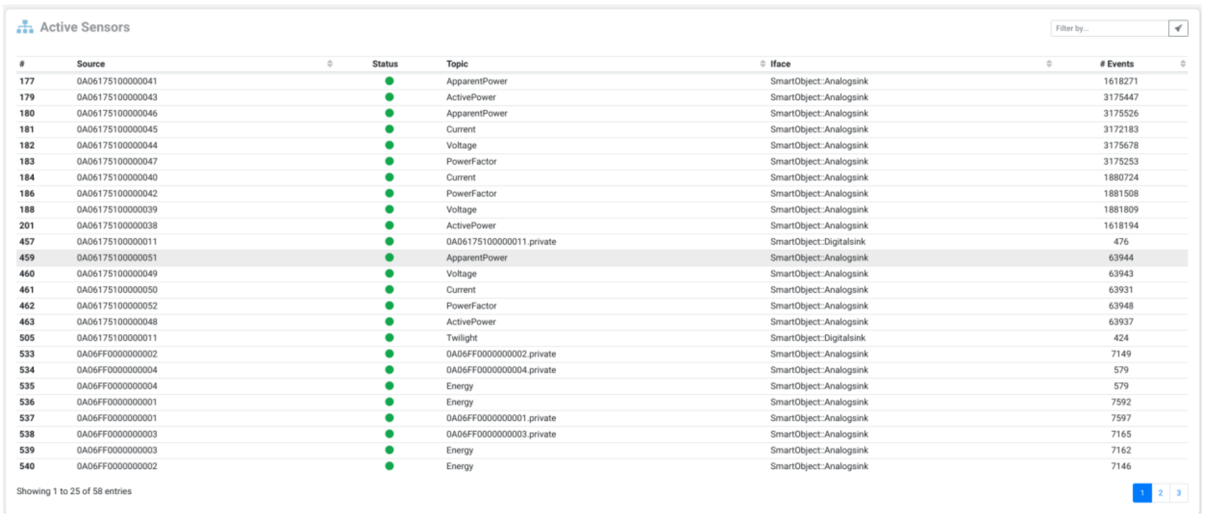
As you can appreciate, we define the endpoint of Citisim instance where we are going to inject the information (line 1), then we define bidir property just in case we need to through a router/gateway with NAT/PAT enabled (line 2) and after defining URLs of the Wibeee sensors (lines 3 and 4), we associate valid Citisim IDs to each physical magnitude metered (lines 5 and 6).

After this config configuration we can use a class Publisher as template:

```
class Publisher:
    def __init__(self, vars, meta):
        self.vars = vars
        self.meta = meta

    def set_ids(self, ids):
        self.ids = {key: ast.literal_eval(values)
                    if values else {} for key, values in ids.items()}
        self.wib_publishers = {key: {} for key in self.ids.keys()}
        self.create_publishers()

    def set_broker(self, broker):
        self.broker = broker

    def create_publishers(self):
        assert self.wib_publishers, "WARNING: no publishers defined!"
        for key in self.ids.keys():
                self.wib_publishers[key][self.vars[0]] = self.broker.get_publisher(
                source=self.ids[key][self.vars[0]], transducer_type="PowerSensor", meta=self.meta)
                self.wib_publishers[key][self.vars[1]] = self.broker.get_publisher(
                source=self.ids[key][self.vars[1]], transducer_type="PowerSensor", meta=self.meta)
                self.wib_publishers[key][self.vars[2]] = self.broker.get_publisher(
                source=self.ids[key][self.vars[2]], transducer_type="PowerSensor", meta=self.meta)
                self.wib_publishers[key][self.vars[3]] = self.broker.get_publisher(
                source=self.ids[key][self.vars[3]], transducer_type="CurrentSensor", meta=self.meta)
                self.wib_publishers[key][self.vars[4]] = self.broker.get_publisher(
                source=self.ids[key][self.vars[4]], transducer_type="EnergySensor", meta=self.meta)
                self.wib_publishers[key][self.vars[5]] = self.broker.get_publisher(
                    source=self.ids[key][self.vars[5]], transducer_type="VoltageSensor", meta=self.meta)

    def publish(self, values):
        for key in values.keys():
            for var in values[key].keys():
                self.wib_publishers[key][var].publish(values[key][var])
                logging.info("{} [{} {}] {}".format(
                    datetime.now(), key, var, values[key][var]))
            logging.info("")
```

This template takes basically the config data from the config file and creates a publisher by each defined sensor id (*create_publishers* method). The broker.get_publisher is a method provided by libcitisim which creates a publisher object of the type that you indicate in *transducter_type* (which is associated to a specific interface). There are up to 40 different types of trasnducter defined.

After this creation of publisher, each time we want to generate an event in Citisim associated to a physical magnitude we call to publish methdo which publish an event for each magnitude. Wibeee sensors meter several magnitudes (energy, current, power and voltage) so each one of that magnitudes are virtually associated to a sensor in Citisim.



#	Source	Status	Topic	Iface	# Events
177	0A06175100000041	●	ApparentPower	SmartObject::Analogsink	1618271
179	0A06175100000043	●	ActivePower	SmartObject::Analogsink	3175447
180	0A06175100000046	●	ApparentPower	SmartObject::Analogsink	3175526
181	0A06175100000045	●	Current	SmartObject::Analogsink	3172183
182	0A06175100000044	●	Voltage	SmartObject::Analogsink	3175678
183	0A06175100000047	●	PowerFactor	SmartObject::Analogsink	3175253
184	0A06175100000040	●	Current	SmartObject::Analogsink	1880724
186	0A06175100000042	●	PowerFactor	SmartObject::Analogsink	1881508
188	0A06175100000039	●	Voltage	SmartObject::Analogsink	1881809
201	0A06175100000038	●	ActivePower	SmartObject::Analogsink	1618194
457	0A06175100000011	●	0A06175100000011.private	SmartObject::Digitalsink	476
459	0A06175100000051	●	ApparentPower	SmartObject::Analogsink	63944
460	0A06175100000049	●	Voltage	SmartObject::Analogsink	63943
461	0A06175100000050	●	Current	SmartObject::Analogsink	63931
462	0A06175100000052	●	PowerFactor	SmartObject::Analogsink	63948
463	0A06175100000048	●	ActivePower	SmartObject::Analogsink	63937
505	0A06175100000011	●	Twilight	SmartObject::Digitalsink	424
533	0A06FF0000000002	●	0A06FF0000000002.private	SmartObject::Analogsink	7149
534	0A06FF0000000004	●	0A06FF0000000004.private	SmartObject::Analogsink	579
535	0A06FF0000000004	●	Energy	SmartObject::Analogsink	579
536	0A06FF0000000001	●	Energy	SmartObject::Analogsink	7592
537	0A06FF0000000001	●	0A06FF0000000001.private	SmartObject::Analogsink	7597
538	0A06FF0000000003	●	0A06FF0000000003.private	SmartObject::Analogsink	7165
539	0A06FF0000000003	●	Energy	SmartObject::Analogsink	7162
540	0A06FF0000000002	●	Energy	SmartObject::Analogsink	7146

Showing 1 to 25 of 58 entries

Figure 2 Citisim Dashboard

With this class, we only have to connect with the Citisim instance:

```
self.broker = Broker(ic=self.communicator())
```

To indicate which variables are being metered and the properties (e.g sensor position):

```
vars = ["pact", "papt", "fpott", "irmst", "eact", "vrmst"]
meta = {"latitude": "38.997947",
"longitude": "-3.919902",
"altitude": "639.10",
"place": "ARCO Lab ITSI"}
```

And to enter in a loop of reading Wibeee sensor and generating the events associated to that update:

```
while(True):
        values = self.requester.get_average_measures(UPDATE_TIME)
        self.publisher.publish(values)
```

The full code can be checked in the wibee-energy-publisher repository of the project.