Making Sense

Advances and experiments
in participatory sensing

SMART CITIZEN TOOLKIT

D2.3

DELIVERABLE

PROJECT ACRONYM	GRANT AGREEMENT #	PROJECT TITLE
Making Sense	688620	Making Sense

DELIVERABLE REFERENCE NUMBER AND TITLE

D2.3
Documentation on firmware to integrate sensors

Revision: v7.0

AUTHORS

Guillem Camprodon (IAAC)
Victor Barberán (IAAC)
Silvia Puglisi (IAAC)

Project co-funded by the European Commision within the Call H2020 ICT2015 Research and Innovation action

DISSEMINATION LEVEL

✓ **P** **Public**

C Confidential, only for members of the consortium and the Commission Services

REVISION HISTORY

REVISION	DATE	AUTHOR	ORG...	DESCRIPTION
v1.0	18-04-2016	Guillem Camprodon	IAAC	First Draft
v2.0	01-06-2016	Victor Barberán	IAAC	Technical reviewing
v3.0	15-06-2016	Silvia Puglisi Victor Barberán	IAAC	Technical reviewing
v4.0	20-06-2016	Guillem Camprodon	IAAC	Integration of contributions
v5.0	21-06-2016	Emma Pareschi	IAAC	Content review
v6.0	21-06-2016	Mara Balestrini	IAAC	Content editing
v7.0	21-06-2016	Gui Seiz	IAAC	Formatting & Design

STATEMENT OF ORIGINALITY

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

INDEX

INTRODUCTION

Since 2012 the Smart Citizen project has aimed to develop tools to support participatory sensing. The two main outcomes of the project are the Smart Citizen Platform and the Smart Citizen Kit, both designed following an open source approach.

This deliverable covers the possibilities that these two tools offer within the context of Making Sense, which focuses on the co-creation of new sensing technologies, data sense-making interfaces, and participation strategies.

The report is structured in two main sections. First, it describes the Smart Citizen platform through its main gateway, the Smart Citizen API, with special focus on how we can publish and retrieve data using different methods. The second section covers the Smart Citizen Kit and provides details on how to use the current hardware with custom sensors.

Since most of the core technologies documented here are already available on the Smart Citizen project documentation under the docs[1] and developer[2] sections on smartcitizen.me, the document provides specific application examples. The aim is to describe the scenarios where this tools can be used and document real examples that can inspire the communities involved in the Making Sense pilots.

> *Note: This deliverable has its own dedicated Github repository[3] containing all the tools and examples explained below.*

1 Smart Citizen Kit documentation https://docs.smartcitizen.me/

2 Smart Citizen API documentation https://developer.smartcitizen.me/

3 Smart Citizen Toolkit repository https://github.com/fablabbcn/smartcitizen-toolkit

THE SMART CITIZEN API

The Smart Citizen platform is built on top of the Smart Citizen public API. This means that any operation performed by the current platform website is available over the API, allowing anyone to built new UIs or to integrate with other tools.

The following section describes how to use the API to retrieve and post sensors' data to the Smart Citizen Platform. The API can also be used to manage devices and users automatically, a feature that although possibly interesting for specific pilot deployments, will not be covered in this deliverable.

1
RETRIEVING DEVICES' DATA

Any user can access the API to retrieve latest or historical data about any device on the platform, even when he does not own it. The API supports HTTPS Rest[4], Websockets[5] for real time services and CSV for device historical data.

The following use cases describe possible scenarios that a community might want to experiment with as part of a Making Sense pilot: Trigger notifications, Custom dashboards, Talk to the world and Data on spreadsheets. They focus on giving participants feedback about the data produced by the sensors towards supporting participation engagement and to help make sense of the data.

4 The Smart Citizen API Documentation http://developer.smartcitizen.me/

5 The Smart Citizen Real Time API http://developer.smartcitizen.me/#real-time

1.1 Trigger notifications

In many cases we might find the need to trigger a notification when a certain event occurs on a Smart Citizen Kit. This might imply sending an email to the user or publishing a tweet.

Node-RED[6] is an open-source visual tool that enabled the wiring of hardware devices, APIs and online services. The tool can be easily installed[7] on any local computer or it can be used directly on the Smart Citizen infrastructure[8].

EXAMPLE

Notify the participants by twitter or email when battery is low

In this example we use the Node Red timmer and http request modules to query the Smart Citizen end-point of the device we want to monitor.

Every time we receive a new response the API data payload in JSON is send to a function node, were a small Javascript snippet check the battery level on the received requests.

In case the battery level is decreasing and is below 20% the module returns a new message explaining the battery is low in a user friendly way. When the message is present the Trigger alert module post a new message using the Twitter API. We can easily direct the requests to other modules as the Mail module for emails, another HTTP module to trigger another REST API or the USB/Serial module to talk to a physically connected Arduino UNO board.[9]

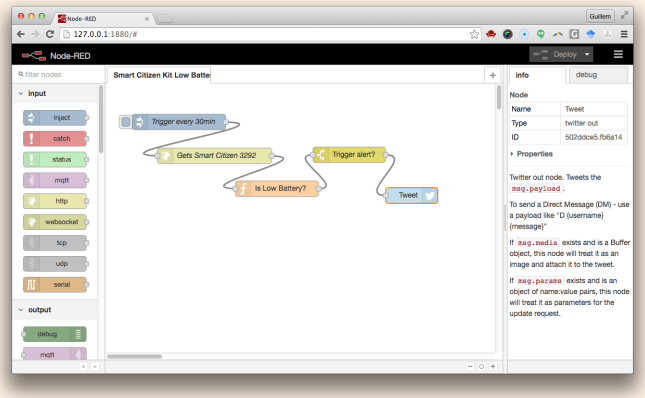
(Example continued in following page...)

6 NodeRed http://nodered.org/

7 NodeRed Installation http://nodered.org/docs/getting-started/installation

8 Node Red testbed at Smart Citizen http://tools.smartcitizen.me/nodered

9 Arduino is an open hardware and software microcontroller kit for building digital devices and interactive objects that can sense and control physical devices https://www.arduino.cc/



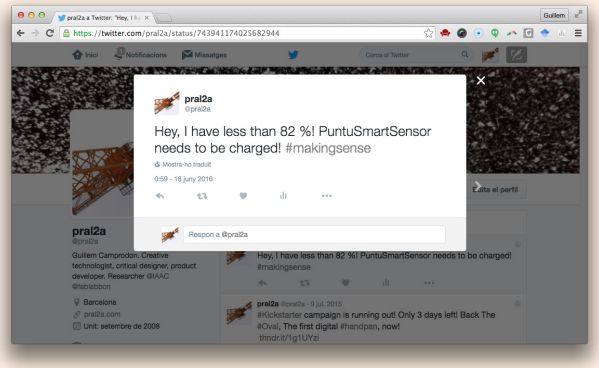


Fig 1 and 2. The Node Rest interface and tweet triggered after the low battery alert

The following example can be downloaded on the Smart Citizen Toolkit repository under node-red and loaded on any Node Red instance.

1.2 Create a custom dashboard

When working on deployments that involve multiple devices a community might face the need to create their own page where the sensors' data is updated on real time.

This can help to look at data from different spots simultaneously and also to create a sense of community among the devices' owners. This feature can be easily built using Freeboards[10], an on-line[11] free visual tool that supports the creation of dashboards.

10 http://freeboard.io/

11 https://freeboard.io/board/ldMIwl

EXAMPLE

The city noise dashboard

The following example shows a dashboard where noise data from three Smart Citizen Kits that are placed in the city of Manchester, are displayed in real-time. The dashboard is publicly accessible on-line.

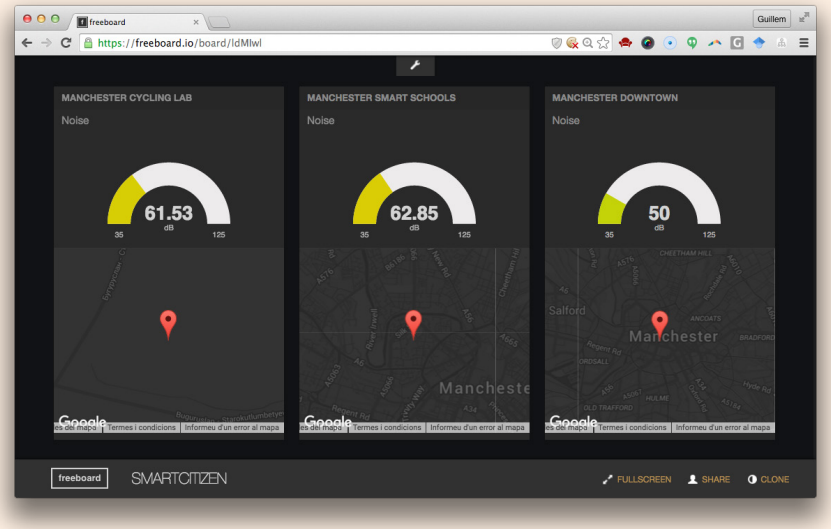


Fig 3. The Freeboard Noise in Manchester dashboard

The following example can be downloaded on the Smart Citizen Toolkit repository under freeboard and loaded on Freeboards as a boilerplate example.

1.3 Talk to the world

Due to their unobtrusive nature, sensor technologies like Smart Citizen may easily blend in the background of users' attention.

To bring the sensed data back to the surface and support sensemaking and awareness processes, it is possible to use the SCKs' data to trigger actions on the physical environment.

The Raspberry Pi platform, a series of credit card-sized low-cost (less than 30€) single-board computers capable of running Linux, is the perfect companion for building this kind of tools. It is also a suitable tool to engage people with coding, creating new internet of things (IoT) and physical computing applications.

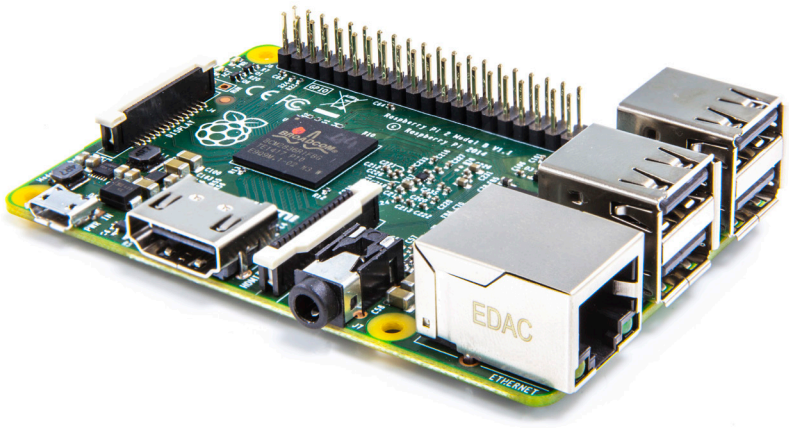


Fig 4. A Raspberry Pi board running Linux for less than 30€

EXAMPLE

The sensor status lights

This example presents a small Python script that can turn two lights based on the real-time temperature data from a remote sensor on the Smart Citizen platform.

We will implement a simple logic: When temperature on the remote sensor reaches 25 degrees then turn the first light on. When temperature is below 25 degrees turn the first light off and then turn the other light on.

We will use the Raspberry Pi GPIOs (General Purpose Input Outputs) to connect to LED's that represent the status of our sensor.

We will need to wire the two LED's following the schematic below. Once we have the Raspberry Pi running and connected to the internet we will need to save the Python script on the desktop, open the Terminal app and run pi@raspberrypi ~ $ cd Desktop && sudo python smartcitizen-led.py

```
# Smart Citizen Examples for the Raspberry Pi
#
# http://smartcitizen.me
#
# Trigger 2 LEDs depending on the temperature
    # For more information on the LEDs connection check: https://learn.sparkfun.com/tutorials/
    raspberry-gpio
# For more information on the SmartCitizen API check: http://developer.smartcitizen.me
#

import RPi.GPIO as GPIO
import json, requests, time

GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)

while True:
        r = requests.get('https://api.smartcitizen.me/v0/devices/3292')
        data = json.loads(r.text)
        for sensor in data['data']['sensors']:
                if sensor['description'] == 'Temperature': #CO, NO2...
                        print sensor['value']
                        if sensor['value'] > 25:
                                print 'LED ON'
                                GPIO.output(18, GPIO.HIGH)
                                GPIO.output(23, GPIO.LOW)
                        else:
                                print 'LED OFF'
                                GPIO.output(18, GPIO.LOW)
                                GPIO.output(23, GPIO.HIGH)
        time.sleep(15) #Update every 15 seconds
```

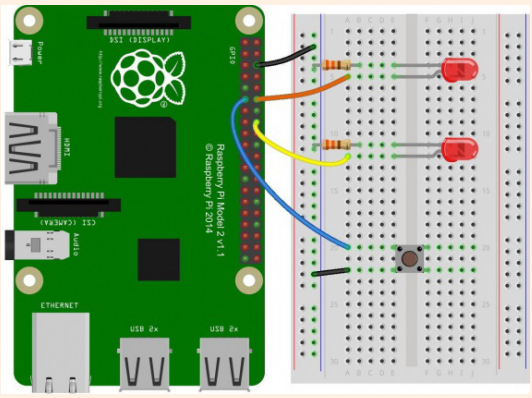


Fig 5. The Raspberry Pi wiring schematic

The above example can be downloaded on the Smart Citizen Toolkit repository under Raspberry Pi.

1.4 Analyzing data using spreadsheets

A way to introduce participants into examining the sense data is through the use of spreadsheets software such as Open Office, Microsoft Excel or Google Spreadsheets, to mention the most broadly used.

The Smart Citizen Platform supports the request of a CSV (Comma Separated Values) file with the data of any device in the platform. Upon request, users receive an email with the processed CSV within less than one minute. This can also be triggered over the API for users to receive data over email periodically[12].

The following example uses Google Spreadsheet[13] since it is free and has multiuser support, allowing participants to add and share comments on data. For CSV containing more than 5000 cells we suggest using Google Fusion Tables[14] instead, also available within Google Docs suite.

EXAMPLE

Looking at data using Google Spreadsheets

Participants can upload their CSV files by simply using the File/Import option in Google Spreadsheets and no special settings are required.

People can then apply standard mathematical operations into the data (e.g. averages) as well as profit from the commenting tools to add annotations and mention other participants.

12 API Request for CSV files http://developer.smartcitizen.me/#csv-archive-of-readings

13 Google Spreadsheets https://www.google.com/sheets/about/

14 Google Fusion Tables https://support.google.com/fusiontables/answer/2571232

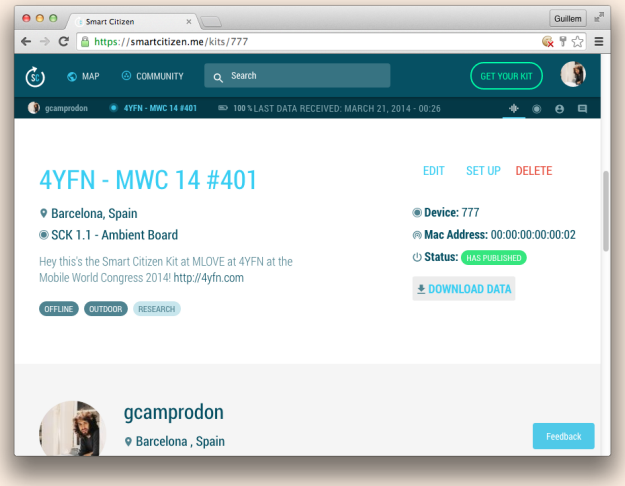


Fig 5

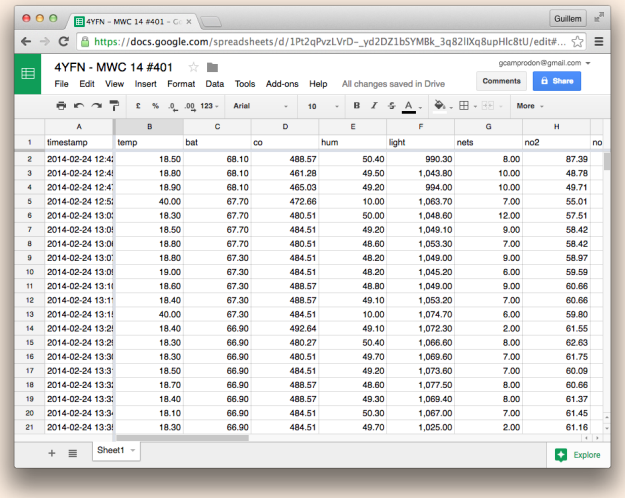


Fig 6

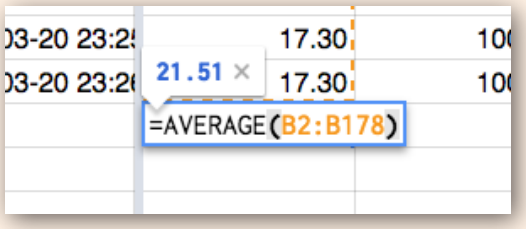


Fig 7

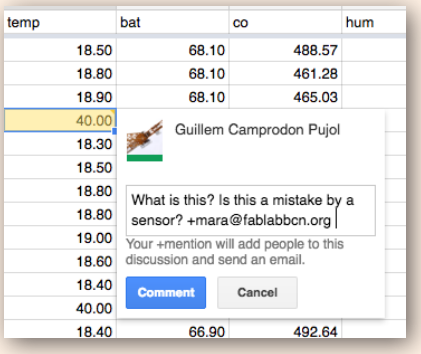


Fig 8

Fig 5, 6, 7 and 8. The download data as CSV option on the Smart Citizen Platform and the later usage in Google Spreadsheets

2
PUBLISHING DATA USING CUSTOM DEVICES

The Smart Citizen Platform supports data from any sensor that has a numerical digital output.

The current Platform supports data ingestion over the same API, which has been already described in a previous section. MQTT[15] support is currently under tests and will be finalised during the upcoming months as part of the release of the Smart Citizen Kit 1.5 version.

2.1 Device blueprints

The Smart Citizen API supports other devices to publish data to the platform by previously agreeing with the Smart Citizen terms and conditions.

For each device type that we intend to add, we would need to create a device blueprint. A device blueprint defines the sensors and the metrics that your devices will have. This will include the hardware details of your sensors and the kind of data that will be published to the platform. Custom calibration formulas to be applied to the data when processed in the platform can be also added.

Once a device blueprint is added to the platform, any user can create as many devices as he likes and publish data to them following the standard Smart Citizen API. It is important to note that Device Blueprint currently cannot be created by users and should be requested by contacting support@smartcitizen.me.

15 MQTT is a machine-to-machine connectivity protocol for constrained devices http://mqtt.org/

The minimal Device Blueprint includes all the necessary data that a user might provide in order to create a Kit[16]. It is composed of Components[17] and those can reuse existing Sensors[18] and Measurements[19] definitions. Sensors define the hardware or software component that records the data. Measurements are descriptions of what sensors are recording.

Blueprints can be shared across many devices or can be tailored per device in order to provide dedicated calibration formulas per individual sensor. This is achieved with the Components binding.

The following example shows a basic Device Blueprint in JSON. This is the minimum of information that a blueprint needs:

EXAMPLE

```
{
   “name”: “The Frog”,
“description”: “Custom Arduino Humidity Sensor”,
   “slug”: “ms:0,5”,
   “components”: [{
          “map”: “hum”,
          “equation”: “(125.0 / 65536.0  * x) + 7”,
          “sensor”: {
                 “name”: “HPP828E031”,
                 “description”: “Humidity”,
                 “unit”: “%”,
                 “measurement”: {
                        “name”: “relative humidity”,
                        “description”: “Relative humidity is a measure...”
                 }
          }
   }]
}
```

16 Kits documentation http://developer.smartcitizen.me/#kits

17 Components documentation http://developer.smartcitizen.me/#components

18 Sensors documentation http://developer.smartcitizen.me/#sensors

19 Measurements documentation http://developer.smartcitizen.me/#measurements

The following examples expand the previous Device Blueprint with the complete data model:

EXAMPLE


```
{
   “id”: 10,
   “uuid”: “056e452d-41c4-436d-a640-2157a278037d”,
   “slug”: “ms:0,5”,
   “name”: “The Frog”,
   “description”: “Custom Arduino Humidity Sensor”,
   “created_at”: “2016-06-18T16:25:02Z”,
   “updated_at”: “2016-06-18T16:25:02Z”,
   “components”: [{
          “id”: 35,
          “uuid”: “22da9377-5151-4547-a71b-6fd8958e1330”,
          “equation”: “(125.0 / 65536.0  * x) + 7”,
          “map”: “hum”,
          “sensor”: {
                 “id”: 13,
                 “uuid”: “1c19ae8f-b995-460f-87a3-a9d0c140abfb”,
                 “parent_id”: 19,
                 “name”: “HPP828E031”,
                 “description”: “Humidity”,
                 “unit”: “%”,
                 “created_at”: “2015-02-02T18:24:30Z”,
                 “updated_at”: “2015-07-05T19:54:54Z”,
                 “measurement”: {
                        “id”: 2,
                        “uuid”: “9cbbd396-5bd3-44be-adc0-7ffba778072d”,
                        “name”: “relative humidity”,
                        “description”: “Relative humidity is a measure of the amount of
moisture in the air relative to the total amount of moisture the air can hold. For instance,
if the relative humidity was 50%, then the air is only half saturated with moisture.”
                 }
          }
   }]
}
```

2.2 Post readings

Posting sensor data is done using the defaults Smart Citizen REST API as a simple POST operation as Device Post Readings[20]:

POST https://api.smartcitizen.me/v0/devices/:device_id/readings

PARAMETER	EXAMPLE	REQUIRED?	DESCRIPTION
REQUEST PARAMETERS			
data array		✓	The data payload
DATA PARAMETERS			
recorded_at datetime	2015-07-20 00:00:00 UTC	✓	The time when the reading took place
sensors array			The sensors objects
SENSOR PARAMETERS			
id integer	12	✓	The id
id string	temp		Instead of the id we can use a hash as defined on the Blueprint
value float	22	✓	The value of the sensor (can be an integer or float)

20 Documentation for device post readings http://developer.smartcitizen.me/#post-readings

Example request

```
POST
https://api.smartcitizen.me/v0/devices/1816/readings?access_
token=XEGwy6BsEybbz3BjYxemxfTQcHjAAJ1s3vJkemhdQ45Cq4hvBM7pNlrY48SUjCfai

    {
      “data”: [{
        “recorded_at”: “2016-06-08 10:30:00”,
        “sensors”: [{
          “id”: “temp”,
          “value”: 21
        }]
      }]
    }
```

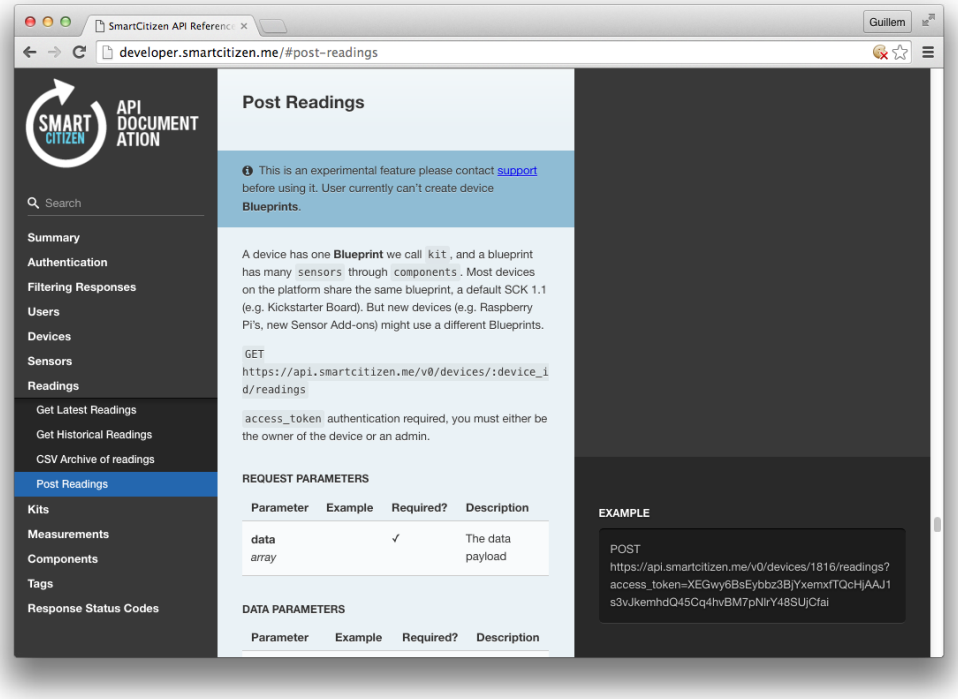


Fig 9. The Smart Citizen API documentation

Extended documentation can be found on the on-line Developers documentation.

2.3 Node.js library

Due to the widespread use of Javascript to develop interfaces and node.js to develop network services, we provide a simple library in order to simplify how to push data to the platform as a standard NPM[21] package.

On any shell running node.js and npm, the module can be installed like

```
$ npm install smartcitizen
```

Here is a simple data post using the module

```
var SmartCitizen = require('smartcitizen')

var smartcitizen = new SmartCitizen({
    id: 8909, // Your device id
    token: 'F1XZt67EG9yaOE5k6yfHhgVQsXl4SbsDcE4ZBc4VMGtsTrOeBri7VjwcqZONWqDVE' //
Your token
});

smartcitizen.push({
    recorded_at: new Date(),
    sensors: [{
        id: 'noise',
        value: 59
    }]
})
```

21 Smart Citizen NPM package page https://www.npmjs.com/package/smartcitizen.

2.4 Authentication

Authentication when publishing data is currently achieved via a unique private user key. Users can request it on the user profile page at:

`https://smartcitizen.me/profile/users`

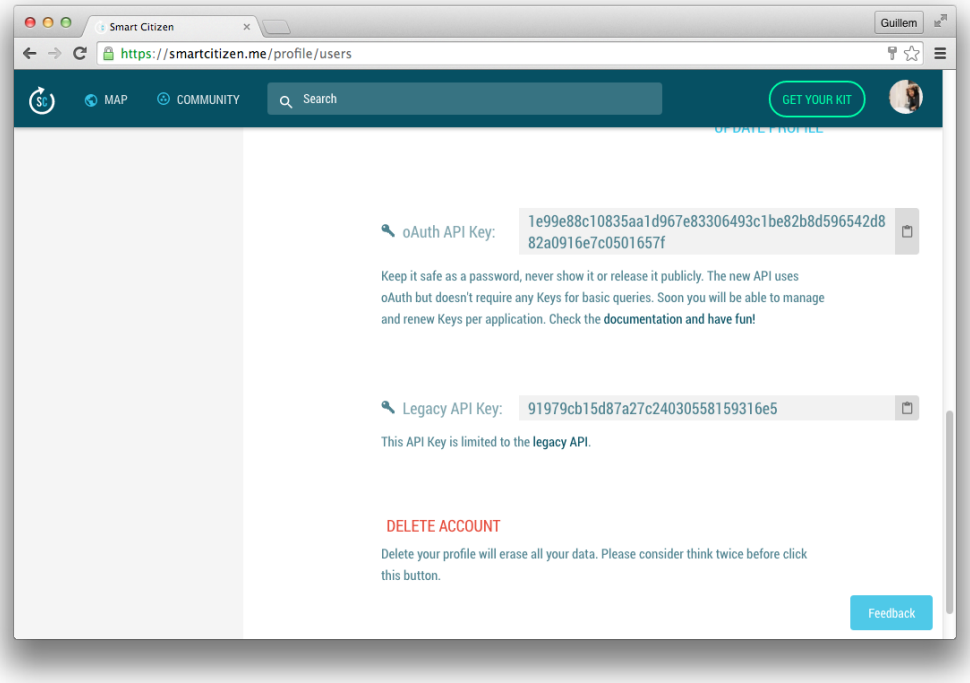


Fig 10. The Smart Citizen Platform user profile with the user Keys

Support for oAuth[22] application key management is already planned and a beta version can be tested here `https://id.smartcitizen.me/oauth/applications`.

22 Authentication documentation http://developer.smartcitizen.me/#authentication

2.5 Adding data from other platforms

There are many scenarios where we can not use the Smart Citizen Kit alone. In many cases we might found other platforms have data we might like to integrate or other sensors we can buy from other manufacturers or develop in our community.

The following use cases: Pulling data, Publishing off-line data and Making our own sensor cover possible scenarios that we found a community might encounter while working on a Making Sense pilot. They are focused on expanding the Smart Citizen Tools by connecting with other existing tools and sensing methodologies.

2.5.1 Pulling data

Some users might want to use data that are already available via other open data platforms, or they might want to integrate an industrial sensor that has its own proprietary data platform.

However many of those platforms have some sort of Webservice or API that are used to retrieve their own data. The use of multiple data sets and platforms can be specially useful to cross reference data. To provide an example, in the next section we show how to integrate data from the Barcelona Sentilo Platform, an official data platform by the city council, with Smart Citizen.

EXAMPLE

Integrating data from the Barcelona Sentilo Platform

We built a simple Node.js sketch to request data at every minute from the Barcelona Sentilo Platform API and publish it into the Smart Citizen platform using the Smart Citizen NPM module.

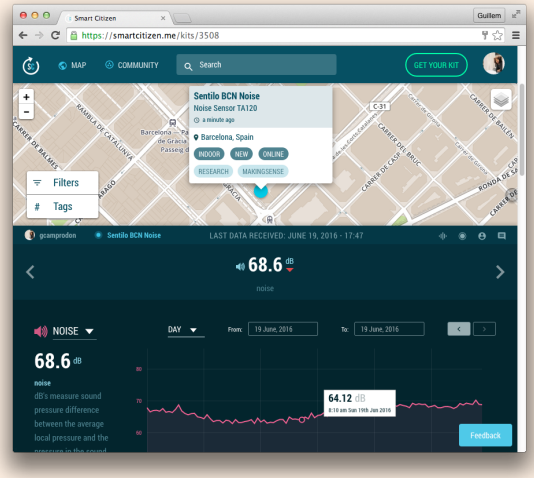
We run the script into our lab Raspberry Pi so it runs continuously and remotely. This kind of integration can be also easily done using the Node Red tool described above.

The script below shows how minimal this kind of integration can be. The source code is available within the Smart Citizen Toolkit repository.

```
// Node.js script for connecting with other APIs
var SmartCitizen = require('smartcitizen'), request = require('request'), moment =
require('moment');

var smartcitizen = new SmartCitizen({
    id: 3508,
    token: 'XEGwy6BsEybbz3BjYxemxfTQcHjAAJ1s3vJkemhdQ45Cq4hvBM7pNlrY48SUjCfai'
});

setInterval(function() {
    request.get({
        json: true,
        url: 'http://connecta.bcn.cat/connecta-catalog-web/admin/sensor/lastOb/CESVA.
TA120-T240427.TA120-T240427-N'
    }, function(err, httpResponse, body) {
        if (body && body.value && body.timestamp) {
            smartcitizen.push({
                recorded_at: moment(body.timestamp, 'DD/MM/YYYYThh:mm:ss'),
                sensors: [{
                    id: 'ta120noise',
                    value: Number(body.value)
                }]
            })
        }
    });
}, 60000);
```



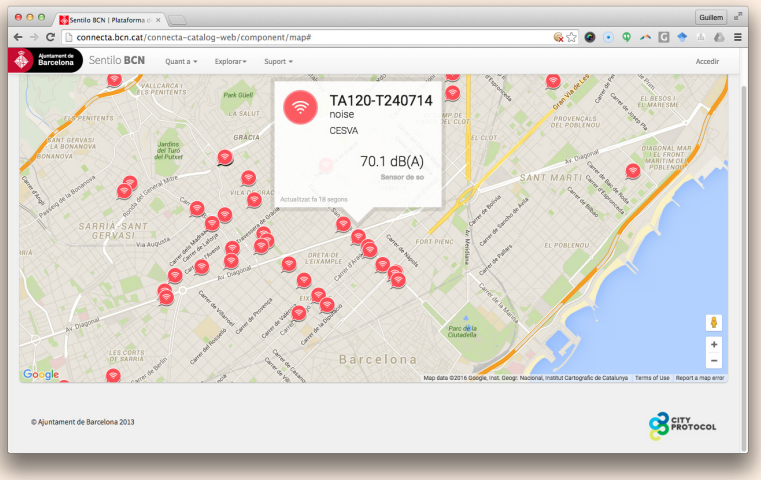


Fig 11 and 12. The Barcelona Sentilo Platform and the sensor once integrated into Smart Citizen[23]

23 The Sentil sensor on the Smart Citizen Platform https://smartcitizen.me/kits/3508

2.5.2 Publishing offline

In certain situations, users might want to employ data capturing tools that can't connect to the internet.

They might be deployed in places where there is no internet connection or they might be completely analog sensors, such as Diffusion Tubes often used to measure air quality.

Nevertheless, the user might hope to visualize the collected data via the Smart Citizen platform. A way to do this is by using Google spreadsheets, which we explain in the next section.

EXAMPLE

Automated data upload using Google spreadsheets

We built a simple Javascript sketch that supports the integration of data from a Google Spreadsheet into the Smart Citizen Platform.

This means that participant can easily entry the analog data on the spreadsheet convenient standard and these data will be automatically added onto the platform device. Each tab on the spreadsheet defines the identifier of each device and users can add as many columns as sensors the devices can support. This service can be enabled for specific pilots. It can also be deployed locally as a simple Node Js app on any Mac or Linux computer. The whole source is available within the Smart Citizen Toolkit repository.

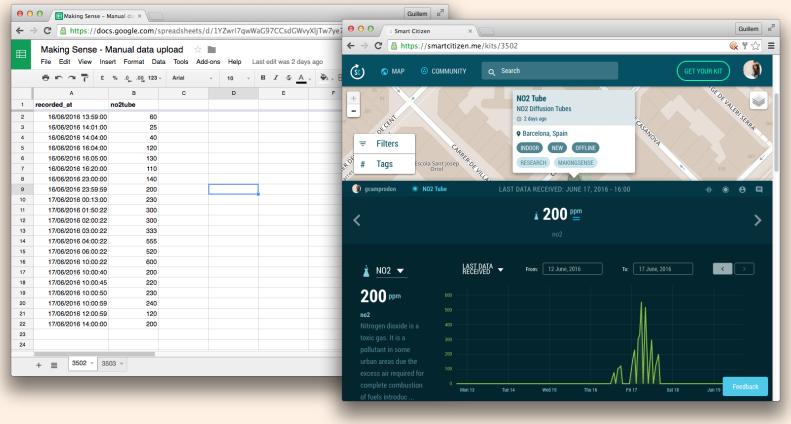


Fig 13 and 14 The Google Spreadsheet syncing data in real time to Smart Citizen[24]

24 https://smartcitizen.me/kits/3502

2.5.3 Making your own sensor

In the process of exploring new ways to sense the environment, communities might find that they would like to appropriate Smart Citizen Kits to prototype their own sensors.

However, it is important to know that for any embedded device to be able to publish directly into the Smart Citizen API it needs to have: internet connectivity and http and ssl support. This includes well known platforms as the new Arduino MKR1000[25] (by using the `WiFi101::ConnectSSL()` class)[26]or the Rasperry Pi.

For devices with just USB support, as the Arduino UNO, Raspberry Pi can be turned into an USB/Serial bridge using tools as Node Red.

EXAMPLE

Measuring street activity using the Raspberry Pi and Processing.org

The cost of cameras is permanently dropping as their computing power increases. This creates new opportunities to develop sensing tools based on Computer Vision strategies, a type of system that was until recently only possible for specialized industries.

Using accessible tools such as Processing and low cost computers like Raspberry Pi we can now build and test novel custom sensors.

The following example uses a Raspberry Pi with the Pi Cam module in order to monitor general activity on a street by performing simple computer vision operations. Data is then published to the Smart Citizen API using the Smart Citizen class for Processing[27].

25 Arduino MKR 1000 https://www.arduino.cc/en/Main/ArduinoMKR1000

26 Arduino WiFI101 Library with HTTPS (SSL) support https://www.arduino.cc/en/Reference/WiFi101ClientConnectSSL

27 Processing is a software sketchbook and a language for learning how to code https://processing.org/

Note this is not aimed at providing a ready to use solution yet showing the potential of the technology. A detailed description of the source is available within the Smart Citizen Toolkit repository.

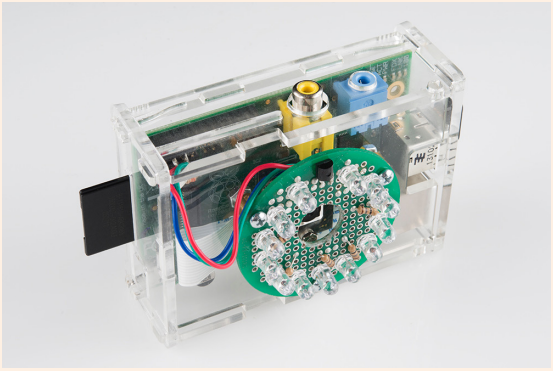


Fig 15 The time-lapse Pi Camera project by Sparkfun[28]

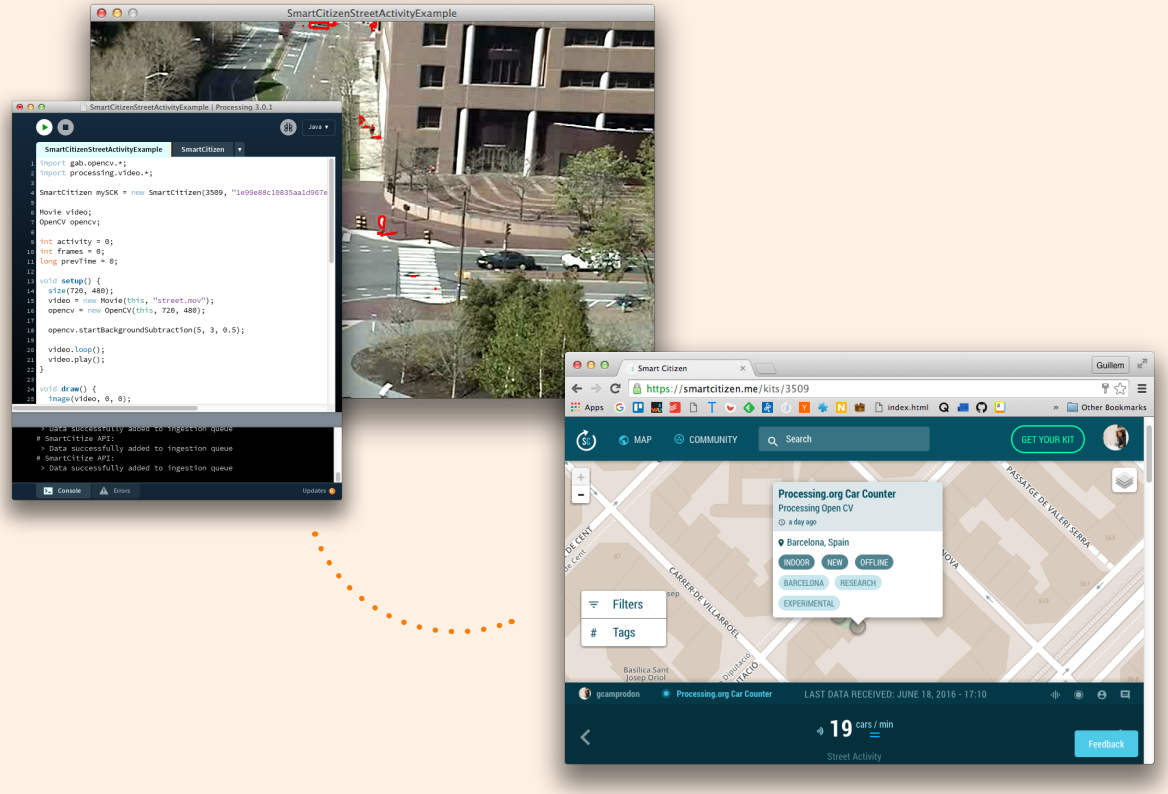


Fig 16 and 17 The Processing sketch pushing data to Smart Citizen[29]

28 Enginursday: Time-lapse with the Raspberry Pi Pt. 2 https://www.sparkfun.com/news/1396

29 https://smartcitizen.me/kits/3509

THE SMART CITIZEN KIT

The Smart Citizen Kit (SCK) is a piece of hardware comprised of a sensors and a data-processing board, a battery and an enclosure.

The first board carries sensors that measure air composition (CO and NO2), temperature, humidity, light intensity and sound levels. Once set up, the device will stream data measured by the sensors over Wi-Fi using the FCC-certified, wireless module on the data-processing board. The device's low power consumption allows for it to be deployed outdoor, e.g. on balconies and windowsills.

Power to the device can be provided by a solar panel and/or a battery. On remote areas data can be also stored on a micro SD card. The latest version can also store up to 1000 reading and post them back when a Wi-Fi network is found.

Fig 18 The Smart Citizen Kit 1.1 with the 3D Printed enclosure

3
HARDWARE DETAILS

SMART CITIZEN KIT	SCK 1.0 (GOTEO BOARD)	SCK 1.1 (KICKSTARTER BOARD)	SCK 1.5 (UPCOMING!)
Public release	04/2013	02/2014	09/2016
DATA BOARD			
MCU	**ATMEGA32U4 (AVR 8-bits)**	**ATMEGA32U4 (AVR 8-bits)**	**SAMD21 (ARM M0+ 32-bits)**
CLOCK	16Mhz	8Mhz	32Mhz
FLASH	32KB	32KB	256KB
RAM	2.5KB	2.5KB	32KB
WIFI	Microchip RN-131 (FCC)	Microchip RN-131 (FCC)	Espressif ESP8266-12E (FCC)
OTHER BUILT-IN PERIPHERALS	Micro SD CARD, RTC (Real TIme Clock), PV + USB LiPo Battery Charger, Add On connector		Micro SD CARD, RTC (Real TIme Clock), PV + USB LiPo Battery Charger, Compass+Accelerometer, Groove Add On connector, RGB LED
FIRMWARE	Repository	Repository	Repository
DESIGN FILES	v1.01	v1.1	v1.5
AMBIENT BOARD			
LIGHT	**PVD-P8001**	**BH1730FVC**	**BH1730FVC**
Type	LDR Analog Light Sensor	Digital Ambient Light Sensor	
Units	%	Lux	
Datasheet	PDV-P8001.pdf	BH-1730FCV.pdf	
Performance		0.008 - 65535 lx (+/- 15%) **	
Firmware	SCKAmbient::getLight();		Under development
TEMP	**DHT22**	**HPP828E031 (SHT21)**	**SHT21**
Type	Digital Temperature and Relative Humidity Sensor		
Units °C	°C	°C	
Datasheet	DHT22.pdf	HTU-21D.pdf	SHT21.pdf
Firmware	SCKAmbient::getDHT22();SCKAmbient::getHumidity();	SCKAmbient::getSHT21(); SCKAmbient::getTemperature();	Under development
Performance		Linearity R^2>0.94 *	Under tests

HUMIDITY	**DHT22**	**HPP828E031 (SHT21)**	**SHT21**
Type	Digital Temperature and Relative Humidity Sensor		
Units	% Rel		
Datasheet	DHT22.pdf	HTU-21D.pdf	SHT21.pdf
Firmware	SCKAmbient::getDHT22();SCKAmbient::getHumidity();	SCKAmbient::getSHT21();SCKAmbient::getHumidity();	Under development
Performance		Linearity R^2>0.97 *	Under tests
NOISE	**POM-3044P-R**	**POM-3044P-R**	**SPU0414HR5H**
Type	Electret microphone with envelop follower sound pressure sensor		New MEMS microphone with envelop follower sound pressure sensor
Units	dB		
Datasheet	POM-3044P-R.pdf		SPU0414HR5H.pdf
Firmware	SCKAmbient::getNOise();		Under development
**Performance*		Range 50dB - 110dB (+/-15%) **	Under tests
CO	**MICS-5525**	**MiCS-4514**	
Type	MOS CO gas sensor	MOS CO and NO^2 gas sensor	
Units	kOhm (ppm)	kOhm (ppm)	
Datasheet	MICS-5525_CO.pdf	MiCS-4514_CO_NO2.pdf	
Firmware	SCKAmbient::getMICS();	SCKAmbient::getMICS();	Under development
Performance		Linearity 0.45 < R^2 < 0.82 *	Under tests
NO^2	**MICS-2710**	**MiCS-4514**	
Type	MOS NO^2 gas sensor	MOS CO and NO^2 gas sensor	
Units	kOhm (ppm)	kOhm (ppm)	
Datasheet	MICS-5525_CO.pdf	MiCS-4514_CO_NO2.pdf	
Firmware	SCKAmbient::getMICS();	SCKAmbient::getMICS();	Under development
Performance		Linearity R^2<0.0 *	Under tests

* South Coast AQMD The correlation coefficient (R^2) is a statistical parameter indicating how well the performance of each sensor compares to that of a Federal Reference or Federal Equivalent Method (FRM and FEM, respectively) instrument. An R^2 approaching the value of 1 reflects a near perfect agreement, whereas a value of 0 indicates a complete lack of correlation

** Internal Smart Citizen Team WIP Evaluation Tests

For more information visit http://docs.smartcitizen.me

4
LICENSES

Licenses and Intellectual Property are always an important topic when expanding an existing platform.

The open source nature of the Smart Citizen Kit provides a robust framework for this work since the entire hardware structure remains under a Creative Commons License (Attribution-nonCommercial-ShareAlike) and the firmware under a GNU General Public License v3.

The firmware license was specifically chosen in order to revert back to the community any modifications or expansions done by other parties. The license requires that developers make new firmware available along with any hardware add-on.

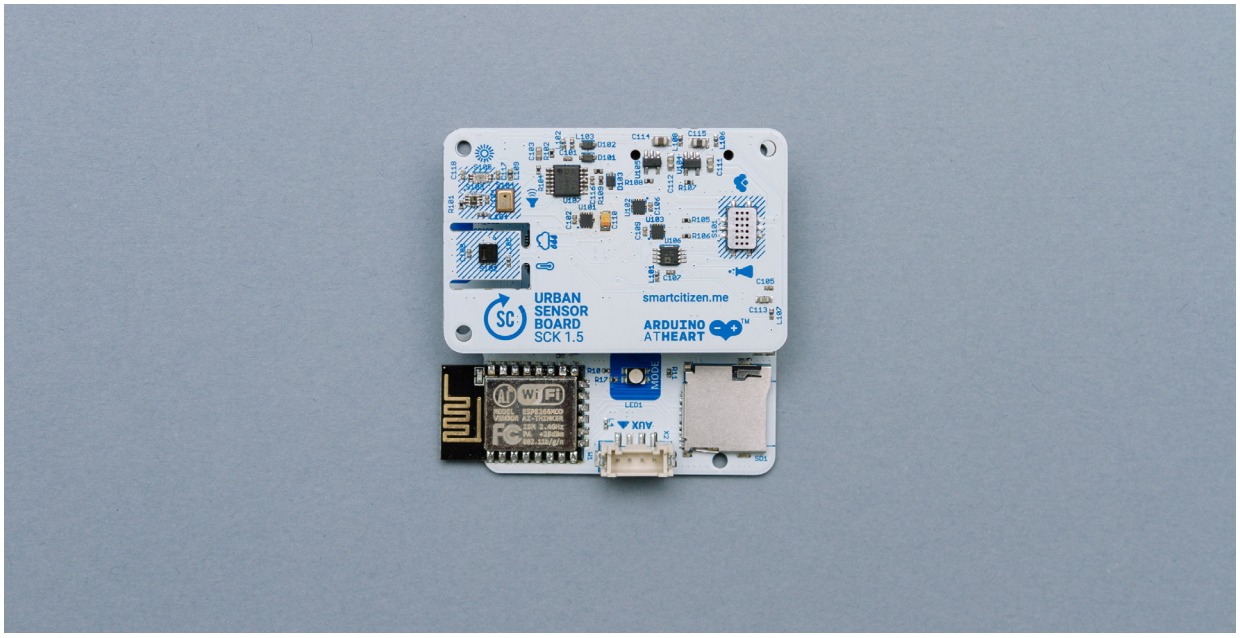


Fig 19 The Smart Citizen Kit 1.5

5
POWER MANAGEMENT

The Smart Citizen can operate on batteries and even using a PV panel.

Even if the hardware design is particularly optimized for low power consumption: we use an ultra low power processor[30] together with low consumption sensors designed for the mobile industry, we must place attention to the power consumption when designing a sensing strategy or when we add new sensors.

There are two critically demanding functionalities: publishing data to the online platform and reading the CO NO2 sensor. This sensor needs to needs to warm up for a long time and consumes close to 60mA seriously affecting the overall battery life.

Smart Citizen Kit 1.1
Power consumption can be controlled by using the online configuration tool to adjust the upload and the sensors reading time in a user friendly UI. We can also perform advance adjustment by editing the Firmware configuration.h SENSOR READINGS - Defaults section in Arduino to change the sensors reading modes.

Smart Citizen 1.5
Sensor can be enabled and disabled by users on the platform, the reading interval can also be set there. The interface is currently under design to support the upcoming Making Sense pilots. This will allow to quickly disable sensor we do not plan to use on a pilot in favour of the battery life.

The standard battery life publishing every minute with all the sensors enabled is 24h, by decreasing the publishing time we can bring it up to 30h. Disabling the CO NO2 sensor we can extend the battery to more than 2 days while publishing every minute or up to a week by reducing the publishing time every hour[31].

30 The Smart Citizen Kit 1.1 uses a low-power Atmel 8-bit AVR RISC microntroller and version 1.5 uses an Atmel SAM based on the ARM® Cortex®-M0+, the most energy-efficient ARM processor available.

31 Data on the report is based on simulations with a 2000mA battery for the Smart Citizen Kit 1.1. A complete report about the Smart Citizen is under development and will be release in https://docs.smartcitizen.me

6
ADDING NEW SENSORS

The Smart Citizen Kit was originally designed to be a modular piece of hardware and software.

However current limitations on the SCK 1.1 processor limited the modularity of the firmware design. Up to the present day this limitations have not stopped 59 people from forking and changing the firmware repository in Github[32].

The new Smart Citizen Kit 1.5, which will be use in Making Sense by late September 2016, has an expanded processor with close to 8 times the firmware space and 16 times more runtime memory. This allows the firmware to be much more modular, therefore simplifying the task of integrating new sensors.

Since the firmware for the Smart Citizen Kit 1.5 is currently under development, dedicated examples and documentation will be published in the upcoming months. At the time we provide the repository where the development process can be followed[33].

32 The Smart Citizen Kit 1.0 and 1.1 firmware https://github.com/fablabbcn/Smart-Citizen-Kit

33 The Smart Citizen Kit 1.5 Repository https://github.com/fablabbcn/Smart-Citizen-Kit-1.

6.1 Designing your own Smart Citizen Kit Sensor Board

The Smart Citizen Kit is comprised of two stacked PCBs, a bottom one dedicated to data processing and communications, and a top one dedicated to the sensors.

The connector between the data board and the sensors board has a standard connector in order to provide power, analog and digital communications (ADC, GPIO, I2C, VCC). Currently a single sensor board is available: the Ambient Board[34].

Anyone can take the PCB design files[35] in Eagle and develop a compatible board. This implies also writing the firmware library to support the sensors taking Smart Citizen Ambient.h[36] library as a reference. This is a good option in the case that a user wants a completely different set of sensors, while keeping a small form factor, and plans to make more than a few units.

34 Check the Hardware specifications on the section above on the Ambient board sensors specifications

35 The Smart Citizen Ambient Board design files https://github.com/fablabbcn/Smart-Citizen-Kit/tree/master/hardware/Kickstarter/v1.1b/Ambient_Board_v1.1b

36 The Ambient.h library is part of the Smart Citizen firmware an is dedicated to sensor management https://github.com/fablabbcn/Smart-Citizen-Kit/blob/master/sck_beta_v0_9/SCKAmbient.h

The diagrams below define the connection sensor boards pin-out for the different Smart Citizen Kits.

SCK 1.0 + SCK 1.1		**SCK 1.5**	
GND	GND	GND	GND
IO3 (10)	IO2 (9)	IO3 (10)	IO2 (9)
IO1 (13)	IO0 (5)	IO1 (13)	IO0 (5)
SCL	SDA	SCL	SDA
S5 (A1)	S4 (A0)	S5 (A1)	S4 (A0)
S3 (A3)	S2 (A2)	S3 (A3)	S2 (A2)
S1 (A5)	S0 (A4)	S1 (A5)	S0 (A4)
VBAT	VBAT	VCC	VCC

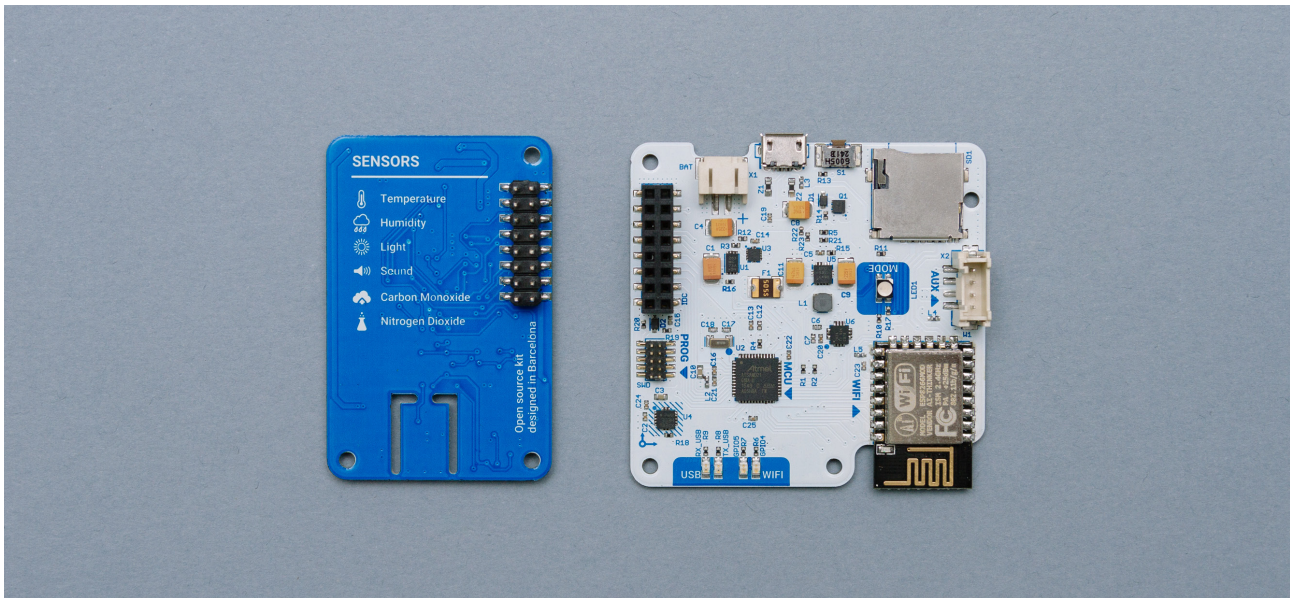


Fig 20 The Smart Citizen Kit Ambient Board and Data Board with their connectors

6.2 Adding sensor (or an actuator) over the expansion port

The Smart Citizen Kit has a built-in expansion port, the Add-on port. This was designed to support I2C sensors or even a slave Arduino board. It can also support Dallas 1-Wire and WS1228. By adding extra accessories we can virtually integrate any sensor as ADC and GPIOs drivers are commonly available as described below.

EXAMPLE

Integrating sensors to the SCK 1.1: The OSBH Smart Citizen Kit

We add two waterproof temperature sensors together with the Ambient Board sensors to measure the temperature inside the beehive.

Since the beehive is located in a remote area we will set the Smart Citizen Kit 1.1 to log on the internal SD Card.

Fig 21 The Smart Citizen Kit expanded for the Open Source Beehives team being set on a Beehive at the Green Fab Lab

1. We integrated two Waterproof Digital Temperature sensor by Maxim, the DS18B20. These are Dallas 1-Wire sensors with a unique ID assigned by the manufacturer.
2. We used the expansion port on top of the SCK 1.1 Sensor Board to connect the 2 sensors.

The expansion port is designed to expand the sensor board by adding new sensors via the common I2C standard. However other protocols are supported via software as on the following case where we use the Dallas 1-Wire protocol, found in many temperature and humidity sensors. The pins have the following configuration:

PIN	PORT	FUNCTION
1	GND	Ground
2	SCL	I2C (by software: 1-WIRE / WS2812)
3	SDA	I2C (by software: 1-WIRE / WS2812)
4	VBAT	Battery Raw Voltage
5	VCC 3.3V	Stable 3.3V Max 500mA

3. The SCK 1.1 firmware needs to be changed in order to support the sensor.
   i. We integrate the Adafruit Library for the WS2812
   ii. The complete firmware implementation is available here on the SCK 1.1 Repository[37]
   iii. In the following example we can see how we momentarily disable the I2C (Wire) bus in order to support other protocols as the Dallas 1-Wire of the WS2812 sensor.

```
#include <OneWire.h>
#include <DallasTemperature.h>

 uint16_t sckGetExtTemp(){
    Wire.end(); // Disable I2C bus on the PIN
    delay(100); // Wait for stability
    extTemperatures.begin(); // Start the WS2812 sensor.
    extTemperatures.getAddress(extTemperatures1, 0); Get the WS2812 address
    extTemperatures.requestTemperatures(); // Send WS2812 temp
    float extTemp = extTemperatures.getTempC(extTemperatures1); // Store it

    #if debuggSCK
      Serial.print("DS18B20 (Ext Sensor): ");
      Serial.print(extTemp);
      Serial.println(" C");
    #endif

    delay(100); // Wait for stability
    Wire.begin(); // Enable the I2C bus on the PIN

    return (int) extTemp*100; // Return
  }
```

37 The Smart Citizen Kit OSBH Firmware branch https://github.com/fablabbcn/Smart-Citizen-Kit/tree/OSBH

4. Since the device is storing data onto the internal SD Card no changes are required on the platform.

6.3 Adding Groove bricks

The new Smart Citizen Kit 1.5 comprises a built in Seedstudio Grove[38] I2C connector as the standard Add-on port.

This means that we can use off-the-shelf sensors from the extensive Groove open hardware sensor library, removing the need to build our own sensor add-ons from scratch.

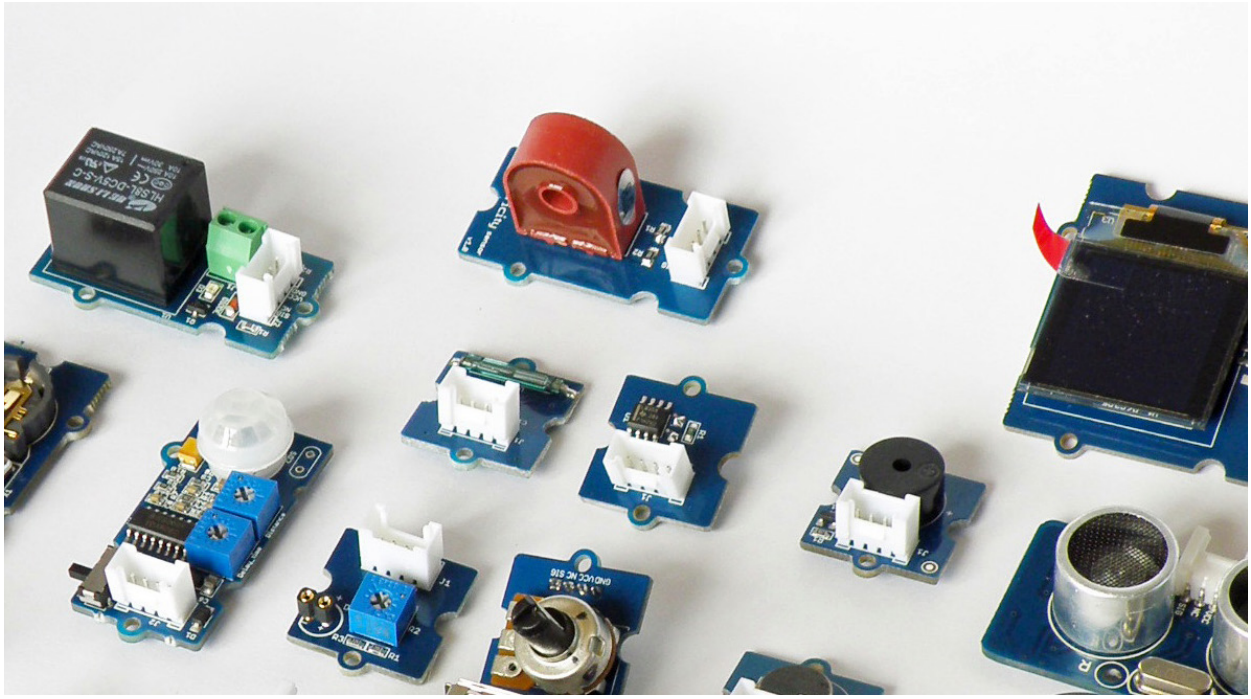


Fig 22 Some of the Seeed Studio Grove sensors and actuators

Seeedstudio Grove sensors natively compatible include:

38 The Seedstudio Grove family documention http://www.seeedstudio.com/wiki/Grove_System

1. Grove - I2C ADC:
A 12-bits high precision Digital to Analog converter

2. Grove - 3-Axis Digital Accelerometer:
A digital accelerometer

3. Grove - 6-Axis Accelerometer Compass:
A digital accelerometer with compass

4. Grove - Chainable RGB LED:
A chainable digital RGB led for status notification

5. Grove - OLED Display 0.96":
A color display for visualizing data and alerts

6. Grove - Motor Driver:
For controlling motor actuators

7. Grove - Color Sensor:
For detecting materials surface colors

8. Grove - I2C Hub:
A hub for connecting multiple Groove I2C ADC modules

By using the Groover - I2C ADC we use the following list of sensors. Using the the Grove - I2C Hub and by changing the Grove I2C ADC address up to 9 sensors can be added.

1. Grove - Water Sensor
2. Grove - Magnetic Switch
3.Grove - Alcohol Sensor
4. Grove - Grove - PH Sensor
5. Grove - Differential Amplifier
6. Grove - Electricity Sensor
7. Grove - Sound Sensor
8. Grove - IR Distance Interrupt
9. Grove - Tilt Switch
10. Grove - Encoder
11. Grove - Moisture Sensor
12. Grove - PIR Motion Sensor
13. Grove - Infrared Reflective Sensor
14. Grove - Digital Light Sensor
15. Grove - Light Sensor
16. Grove - Temperature and Humidity Sensor
17. Grove - Barometer Sensor
18. Grove - Dust Sensor
19. Grove - Air quality
20. Grove - Gas Sensor
21. Grove - Temperature Sensor
22. Grove - Air Quality Sensor
23. Grove - Temperature and Humidity Sensor Pro
24. Grove - Gas Sensor(O_2)
25. Grove - Temp&Humi Sensor(SHT31)
26. Grove - Barometer Sensor(BME280)
27. Grove - HCHO Sensor
28. Grove - Collision_Sensor

Custom I2C add-ons can also be integrated by using the Grove - Screw Terminal

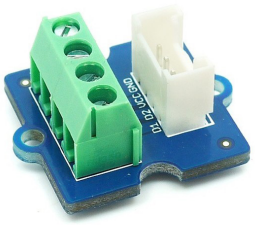


Fig 23 The rew Terminal

Pin	Port	Function
1	GND	Ground
2	VCC	VCC 3.3V 500mA MAX
3	SDA	I2C (by software: 1-WIRE / WS2812)
4	SCL	I2C (by software: 1-WIRE / WS2812)

More examples and documentation will come on the upcoming months when the Smart Citizen Kit 1.5 Firmware will be ready.

EXAMPLE

Adding a Grove Analog Sensor on the SCK 1.5

The Smart Citizen Kit 1.5 firmware supports a single Groover - I2C ADC in a plug-n-play fashion. This means that once the add-on is detected the firmware will automatically start posting the sensor values in mV to the Smart Citizen Platform together with the other sensors.

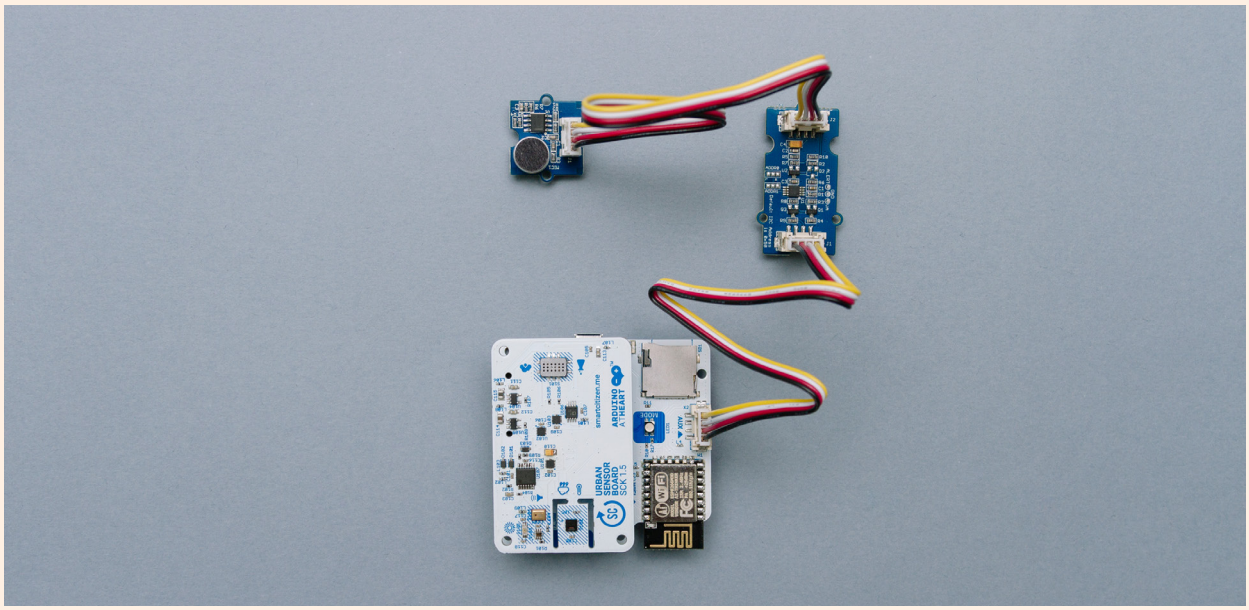


Fig 24 The Smart Citizen Kit 1.5 with the Grove ADC module and the Grobe Microphone module

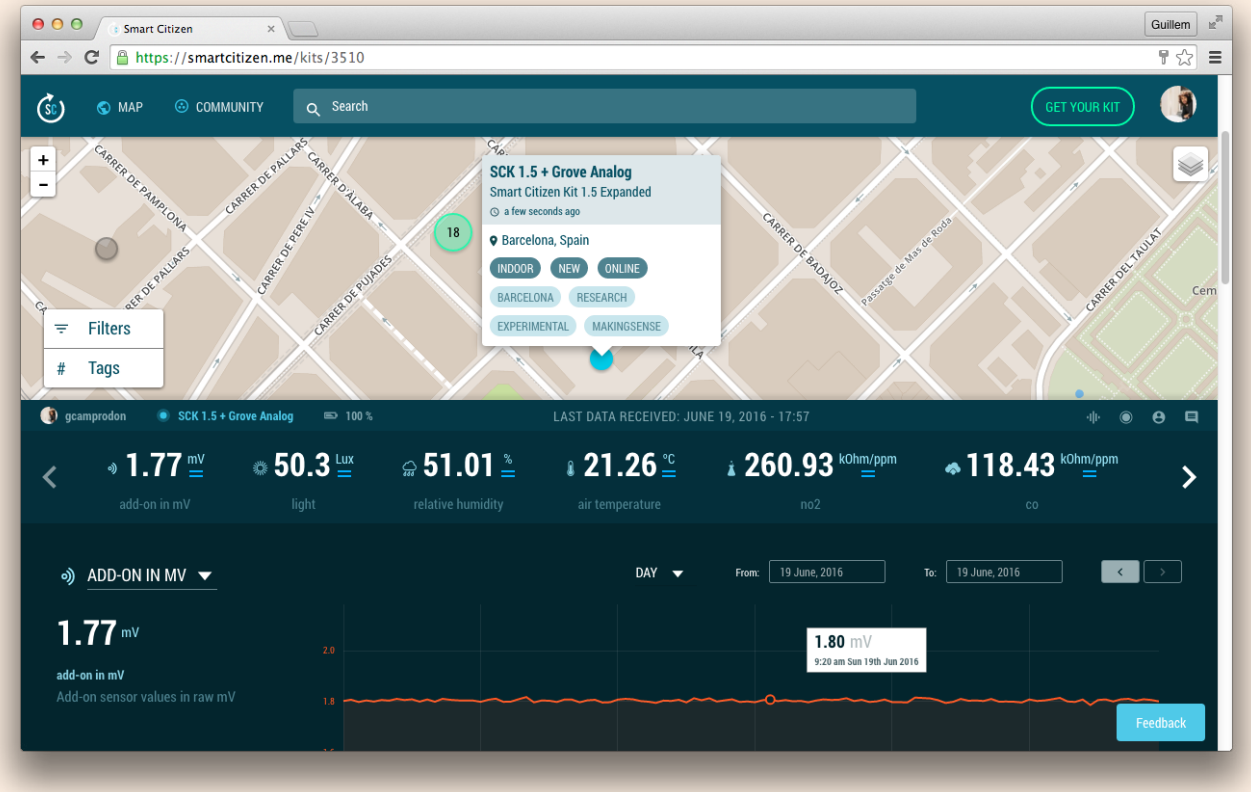


Fig 25 The Smart Citizen Kit 1.5 with an analog sensor add-on on Smart Citizen

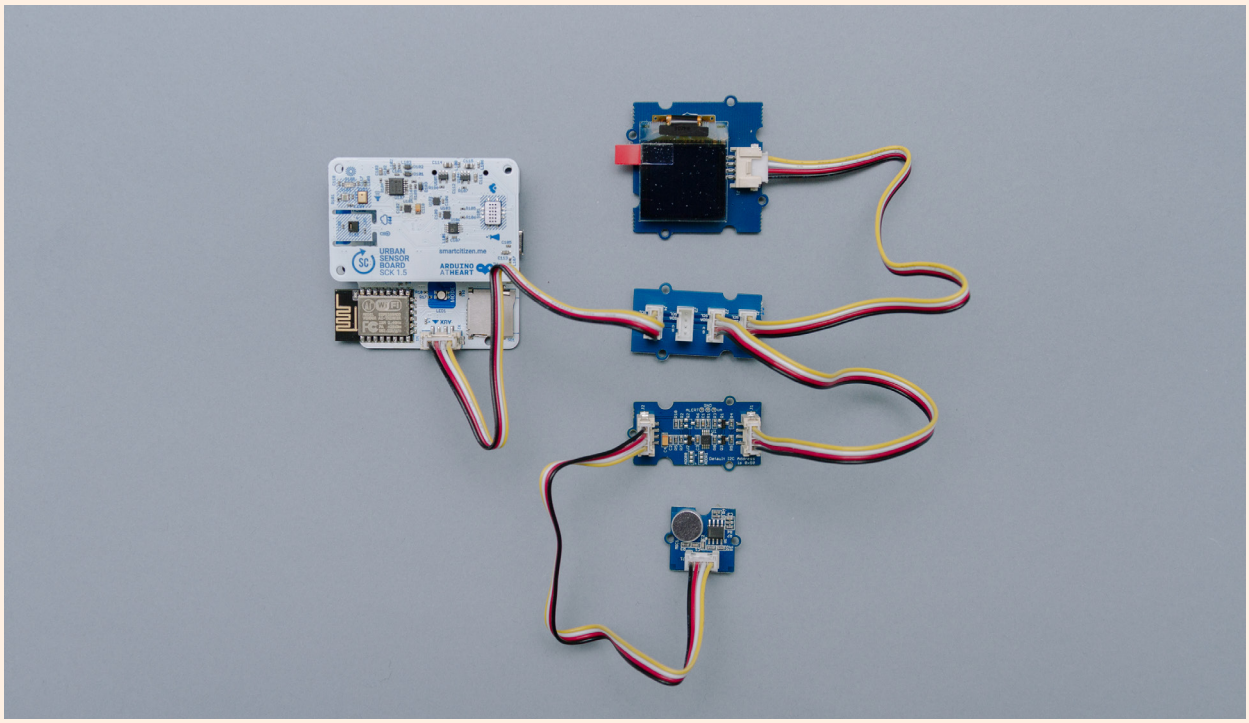
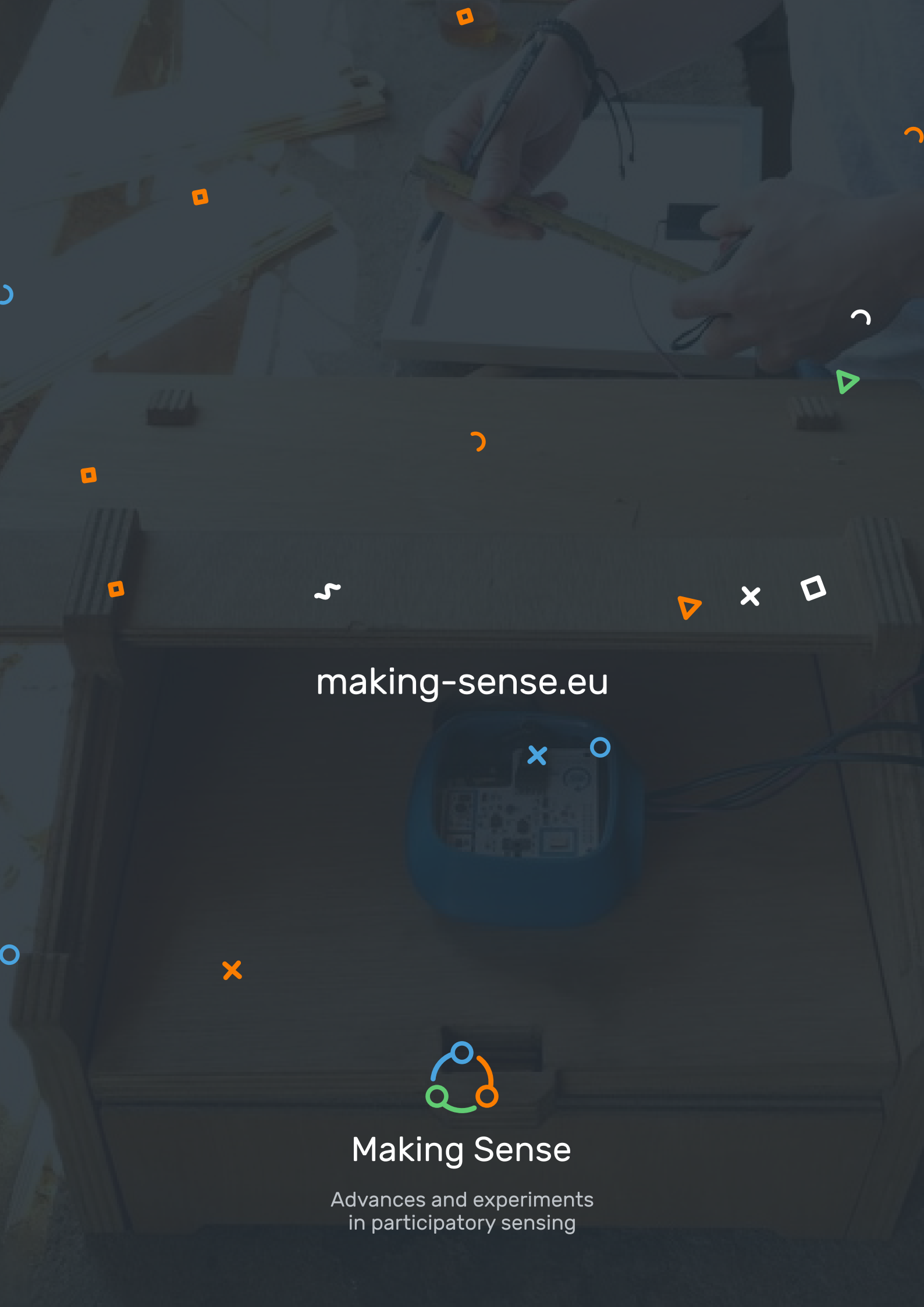


Fig 26 The Smart Citizen Kit 1.5 with multiple Grove modules.
The small display can be used to provide direct feedback to users.

CONCLUSIONS

This deliverable sheds light on the vast amount of possibilities offered by the Smart Citizen tools as key enablers of participatory sensing pilots and experiments.

By providing meaningful examples of novel appropriations and uses we seek to inspire communities to conduct their own experiments and, hopefully, even their own custom tools. The cases here described will be complemented with new informations to be published within the next few months along with the public release of the Smart Citizen Kit 1.5.



making-sense.eu

Making Sense

Advances and experiments
in participatory sensing