Tiny GSM Example for Live Booster

User Manual for arduino mqtt library

Arduino

Table of contents

1. Introduction

1.1. Document purpose

This document is a complete guide to use Tiny GSM example for Live Object on Arduino platform. It is presenting the following:

- Overview
- Getting started
- Detailed Features

2. Overview

2.1. What is Live Objects?

Live Objects is one of the products belonging to Orange Datavenue service suite.

Live Objects is a software suite for IoT / M2M solution integrators offering a set of tools to facilitate the interconnection between **devices** or **connected « things »** and **business applications**.

The main features provided are:

- **Connectivity interfaces** (public and private) to collect data, send command or notification from/to IoT/M2M devices,
- **Device management** (supervision, configuration, resources, firmware, etc.),
- **Message Routing** between devices and business applications,
- **Data Management** and Data Storage with Advanced Search features.

Read Datavenue Live Objects - complete guide to have a full description of services and architecture provided by Live Objects platform.

2.2. Arduino

Arduino is a well known platform for educational purpose. It is also widely used for testing and prototyping projects.

2.3. Tiny GSM

TinyGSM is a small library for GSM Modules. It supports SIM800, SIM900, A6, A7, A20, M590, U201, XBee, ESP8266-AT.

github : https://github.com/vshymanskyy/TinyGSM

This library use the pubsub client

github : https://github.com/knolleary/pubsubclient

2.4. Live Objects IoT examples using iotsoftbox-mqtt library

2.4.1. Introduction

A good way to discover Live Objects features is to use our Live Objects IoT examples.

When running on a development board, the embedded 'basic' application:

- Connects to Datavenue Live Objects Plaftorm, using:
  - an optional secure connection (TLS)
  - the Live Objects mode: Json+Device
- Publishs
  - The current Status/Info
    - `dev/info`
  - The current Configuration Parameters
    - `dev/cfg`
  - The current Resources
    - `dev/res`
  - Data
    - `dev/data`
- Subscribes to Live Objects topics to receive notifications
  - Configuration Parameters update request
    - `dev/cfg/upd`
      - `validate reception by publishing CID on topic dev/cfg`
  - Resource update request
    - `dev/rsc/upd`
      - `validate reception by publishing R_CID on topic dev/rsc/upd/res`
  - Command request
    - `dev/cmd`
      - `validate reception by publishing C_ on topic dev/cmd/res`
- then the application waits for an event:
  - From Live Objects platform to:
    - Update "Configuration Parameters"
    - Update one "Resource": message or image
    - Process a "Command": "SWITCH_LIGHT"
  - From application simulating some data publish operations.
  - And if the connection is lost, restart atthe first step trying to connect again to the Live Objects platform.

2.4.2. Configure workstation

2.4.2.1. Arduino IDE

Using the Arduino IDE allows to abstract the host platform.

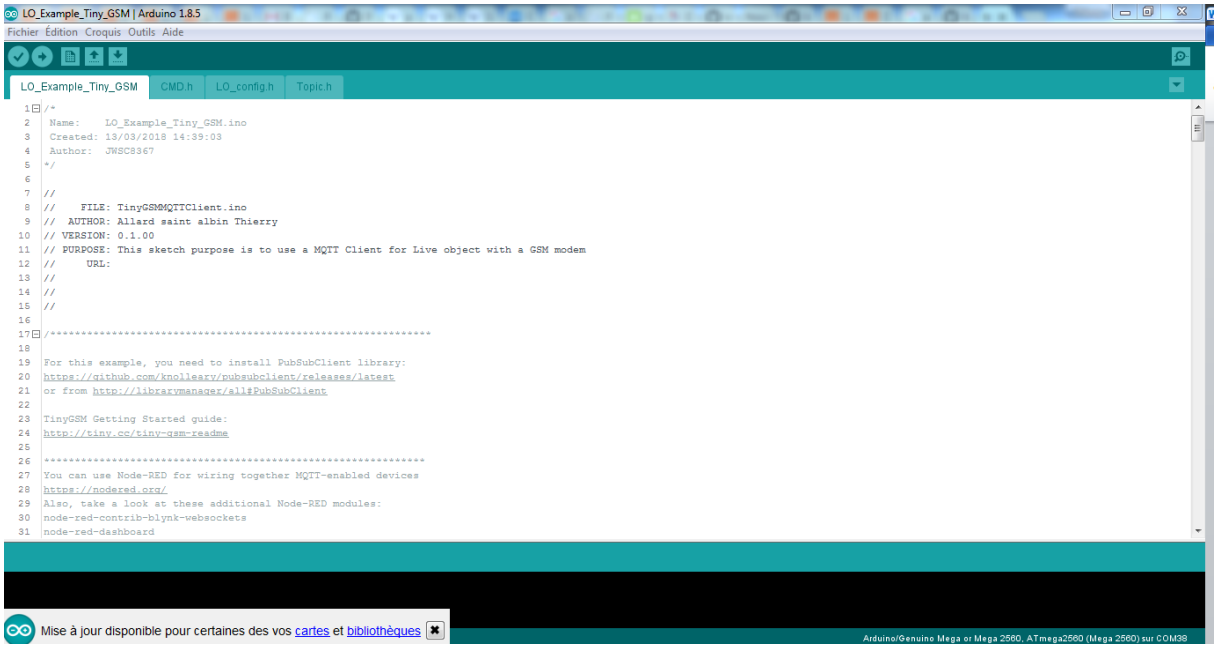
This example is given for a Generic Arduino device as a target. Install the Arduino IDE (tested version: 1.8.1)

Get it from : https://www.arduino.cc/en/Main/Software

1. Launch Arduino IDE
2. Install additional packages/libraries (to use your boards and shields)
    a. TinyGSM : https://github.com/vshymanskyy/TinyGSM
    b. Pubsubclient : https://github.com/knolleary/pubsubclient
    c. ArduinoJson : https://github.com/bblanchon/ArduinoJson

Use the example code in the github

- Example code : https://github.com/IMTOCP/LO_example
- Unzip LO_example-master.zip file
- Rename the folder LO_example-master → "LO_Example_Tiny_GSM"
- copy the folder to your Arduino project folder
    - C:\Users\<username> \Documents\Arduino
- Rename in the folder C:\Users\<username>\Documents\Arduino\LO_Example_Tingy_GSM the file LO_config.h.txt → "LO_config.h."
- Double click on the LO_Example_Tiny_GSM.ino



2.4.2.2. Visual Studio

You will need to install Arduino IDE before visual studio

Install Visual studio community

Get it from : https://www.visualstudio.com/fr/vs/community/

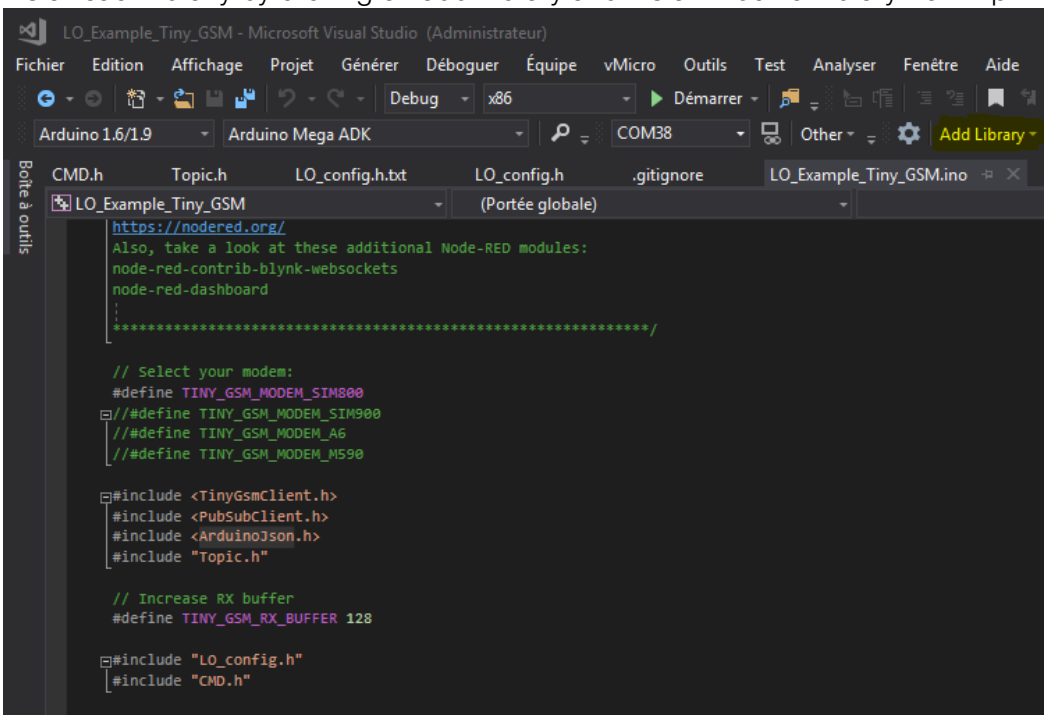
Install visual micro

Get it from : http://www.visualmicro.com/page/Arduino-Visual-Studio-Downloads.aspx

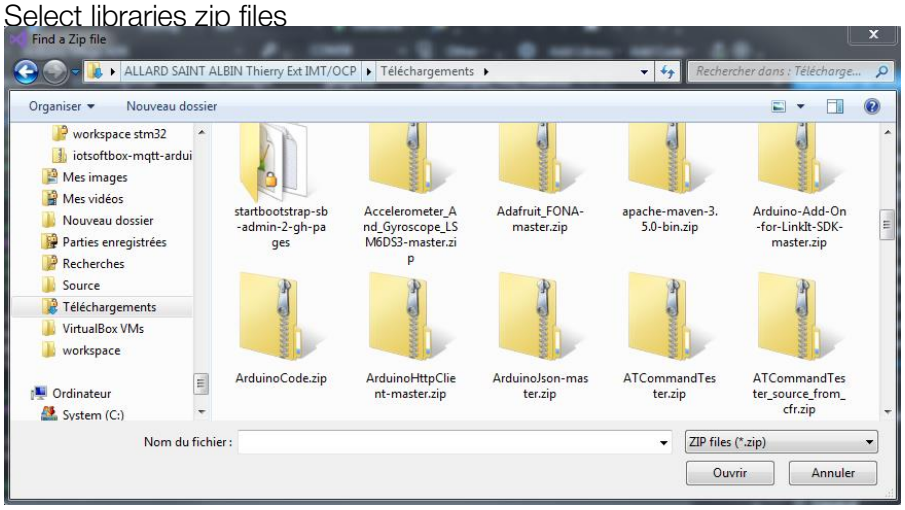
Download libraries

TinyGSM	https://github.com/vshymanskyy/TinyGSM
PubSubClient	https://github.com/knolleary/pubsubclient
ArduinoJSON	https://github.com/bblanchon/ArduinoJson

- Install each librariy by clicking on add Library and Install Arduino Library from zip



- Select libraries zip files



- Click on “ouvrir” and repeat the thoses step for each library

- Copy the folder to your visual project folder if you want
  - C:\Users\<username>\Documents\Visual Studio 2017\Projects
- Double click on LO_Example_Tiny_GSM.sln in the folder LO_example-master
- If ask you save devenv.sln, save it in the project folder
- In the solution explorer, delete LO_config.h in the headers folder
- rename LO_config.h.txt → “LO_config.h” in headers folder

2.4.3. Build

2.4.3.1. Arduino IDE

To build an example in the IDE, just use **Sketch** -> **Verify/Compile.**

2.4.3.2. Visual studio

To build a project click on “générer” then “Générer LO_Example_Tiny_GSM”

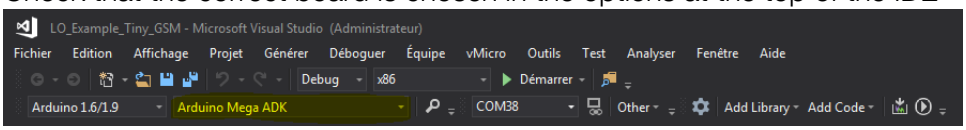
2.4.4. Launch

2.4.4.1. Upload the sample on your Arduino through the Arduino IDE

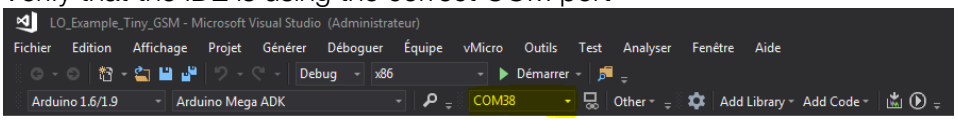
- Connect your board to your computer USB port running the Arduino IDE.
- Check that the correct board is chosen in **Tools** -> **Boards.**
- Verify that the IDE is using the correct COM port (**Tools** -> **Port**).
- To upload a program to your board: **Sketch** -> **Upload**.

2.4.4.2. Upload the sample on your Arduino through the visual studio IDE

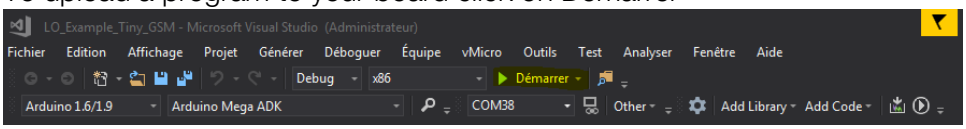
- Connect your board to your computer USB port running the Arduino IDE
- Check that the correct board is chosen in the options at the top of the IDE



- Verify that the IDE is using the correct COM port



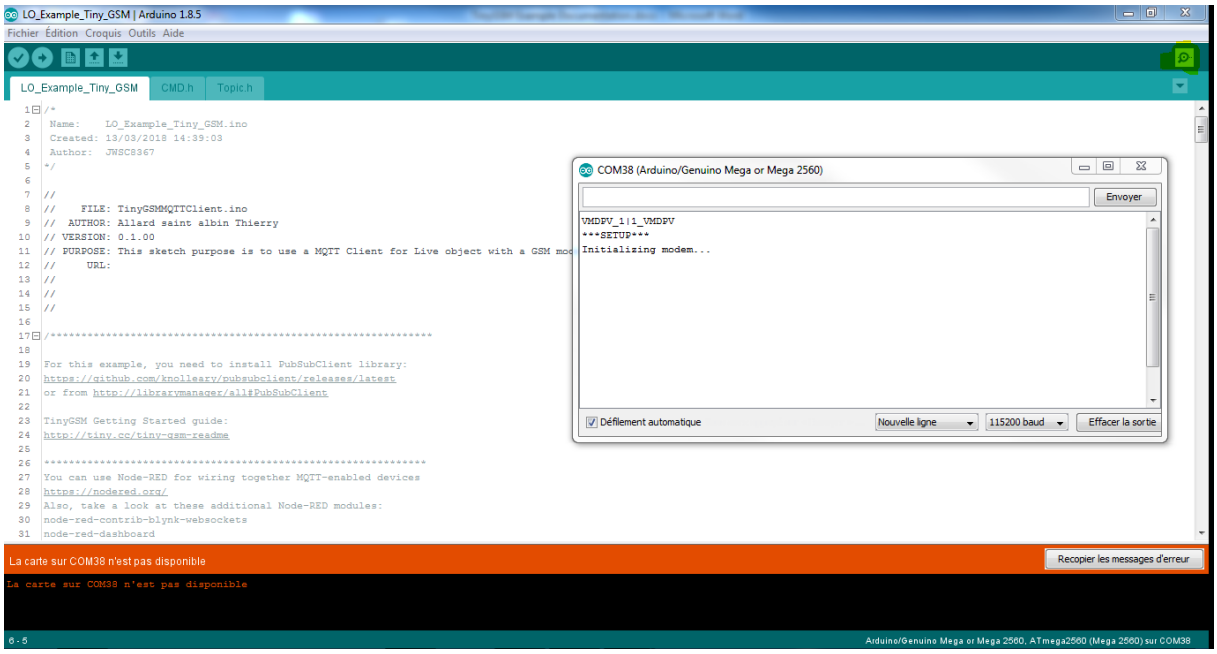
- To upload a program to your board click on Démarrer



2.4.4.3. Excute the application

On Arduino IDE :

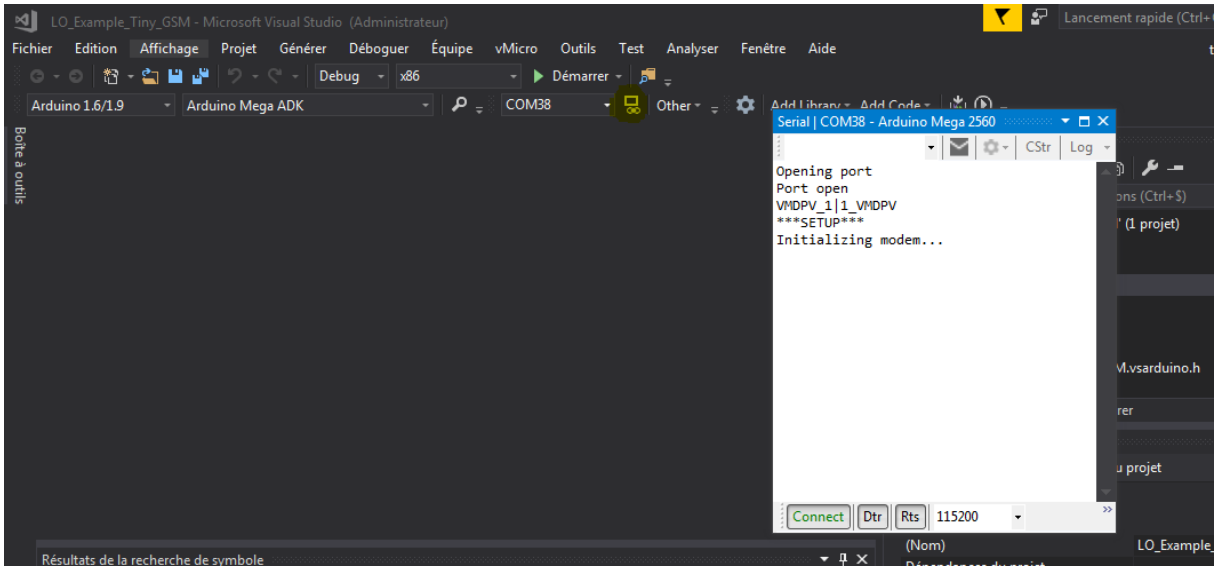
After uploading the board, click on 'Serial Monitor' button  (or CTRL+SHIFT+M).



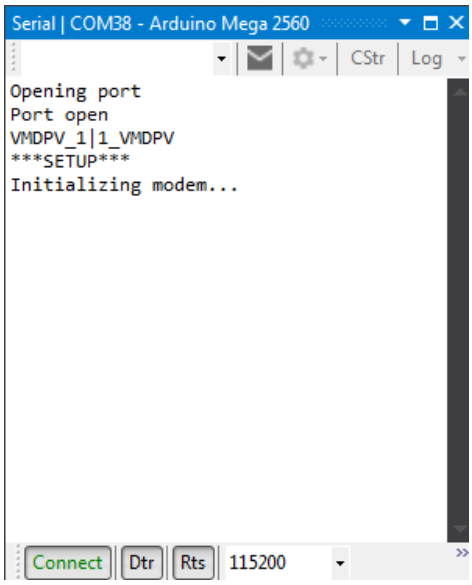
The following window is open:

On Visual studio IDE :

clik on the  on the options on the top of the IDE



the following window is open



2.4.4.4. Application Monitoring/Testing

There is several ways to monitor or/and to test the embedded sample application:

- Go to your Live Objects user account on Live Objects Portal.
- Go to Live Objects Swagger User Interface.
- Serial Terminal is used by embedded sample application to print debug/trace messages.

From Live Objects you can see your board, check the status, check/change configuration parameters, send commands, update resources and more.

To find out more about Live Objects capabilities, see: Live Objects User manual.

3. Detailed Features

3.1. Configuration

The endpoint (Live Objects server) is defined at compile time.

The default values are defined in the iotsoftbox-mqtt library as:

- Server Name: *liveobjects.orange-business.com*
- IP Address: 84.39.42.214 or 84.39.42.208
- TCP Port:
  - 1883 for non SSL connection (without security),
  - 8883 for TLS/SSL connection.
- If TLS is enabled.
  - Public Root Certificate
  - Certificate Common Name 'm2m.orange.com'

Within Datavenue Live Objects platform, the device is identified by its URN:

`urn:lo:nsid:{namespace}:{id}`

The device has to specify:

- *Namespace* identifier, used to avoid conflicts between various families of identifier (ex: device model, identifier class "imei", msisdn", "mac", etc.).
  Should preferably only contain alphanumeric characters (a-z, A-Z, 0-9).
- *Id* (ex: IMEI, serial number, MAC address, etc.)
  Should only contain alphanumeric characters (a-z, A-Z, 0-9) and/or any special characters amongst: - _ | + and must avoid # / !.

To configure your board you need to change variables in the LO_config.h

```
// Your Device id wrote like this
// urn:lo:nsid:<namespace>:<board id>
// example : "urn:lo:nsid: soil_sensors:deviceIMEI1280472603490 "
const char * deviceid = "";


// YOUR GPRS CREDENTIALS
// Leave empty, if missing user or pass
// APN Must be define with this value "object-connected.fr"
const char apn[] = "";
const char user[] = "";
const char pass[] = "";

//BROKER CREDENTIALS
const char* broker = "liveobjects.orange-business.com";
//LO_MDP must be your API Key
const char LO_MDP[] = "";
```

```
const char LO_USER[] = "json+device";
```

3.2. Publish function

```
bool publish(const char * topicArg, JsonObject& root, bool alertUser,
void(*callback)(void))
```

The publish function is used to send data to any topic.

It takes as arguments a topic, a JsonObject, a Boolean.

The callback is only used if the Boolean alertUser is on true.

3.3. Subscribe and receive data

To receive data, parameters and resources we subscribe to topics

- dev/cfg/upd
- dev/cmd
- dev/rsc/upd

```
void subscribeToTopics()
{
	// dev/cfg/upd
	mqtt.subscribe(receiveCfgTopic);

	// dev/cmd
	mqtt.subscribe(cmdTopic);

	// dev/rsc/upd
	mqtt.subscribe(updtRscTopic);
}
```

In the setup function we define a callback function

```
mqtt.setCallback(mqttCallback);
```

This callback is called when the device receive data from any of the topics.

```
void mqttCallback(char* topic, uint8_t * payload, unsigned int len) {


	Serial.print("Message arrived [");
	Serial.print(topic);
	Serial.print("]: ");
	Serial.write(payload, len);
	Serial.println();
	char * buffer = new char[len];
	buffer = (char *)payload;
	StaticJsonBuffer<200> jsonBuffer;
	// Only proceed if incoming message's topic matches
```

```
	if (String(topic) == receiveCfgTopic) { // When receiving message from topic
"dev/cfg/upd"
		Serial.println("Config");
		JsonObject& root = jsonBuffer.parseObject(buffer);
		CID = String((char *)root["cid"]);
		sendConfig();
	}

	if (String(topic) == cmdTopic)   // When receiving message from topic "dev/cmd"
	{
		Serial.println("cmd");
		JsonObject& root = jsonBuffer.parseObject(buffer);
		String tmp_C_CID = root["cid"];
		if (String(C_CID) != tmp_C_CID)
		{
			AlertReceive();
			C_CID = tmp_C_CID;
			String req = root["req"];
			if (req == SWITCH_OFF)
			{
				JsonObject& arg = root["arg"].asObject();;
				int light = arg.get<int>("Light");
				if (light == 0)
					LED_BUILTIN_STATUS = false;
				else
					LED_BUILTIN_STATUS = true;
				digitalWrite(LED_BUILTIN, LED_BUILTIN_STATUS);
			}
			sendReponseCommand();
		}
		else
		{
			Serial.println("CCID don't change");
		}
	}
	if (String(topic) == updtRscTopic) // When receiving message from topic
dev/rsc/upd
	{
		Serial.println("update resource");
		JsonObject& root = jsonBuffer.parseObject(buffer);

		String tmp_R_CID = root["cid"];
		R_CID = tmp_R_CID;
		sendResources();
	}
}
```

Everytime you receive data from those topics. you will need to confirm that you receive the stream by replying with the ID of the stream.

Those functions are :

- sendConfig()
- sendReponseCommand()
- sendResources()

3.4. Status

Status gives information about the device states, i.e. Software version, IP address, connection state, statistic counters.

3.4.1. Push ‘status’ data

Status data is configure and push to on the web in the sendStatus() method.

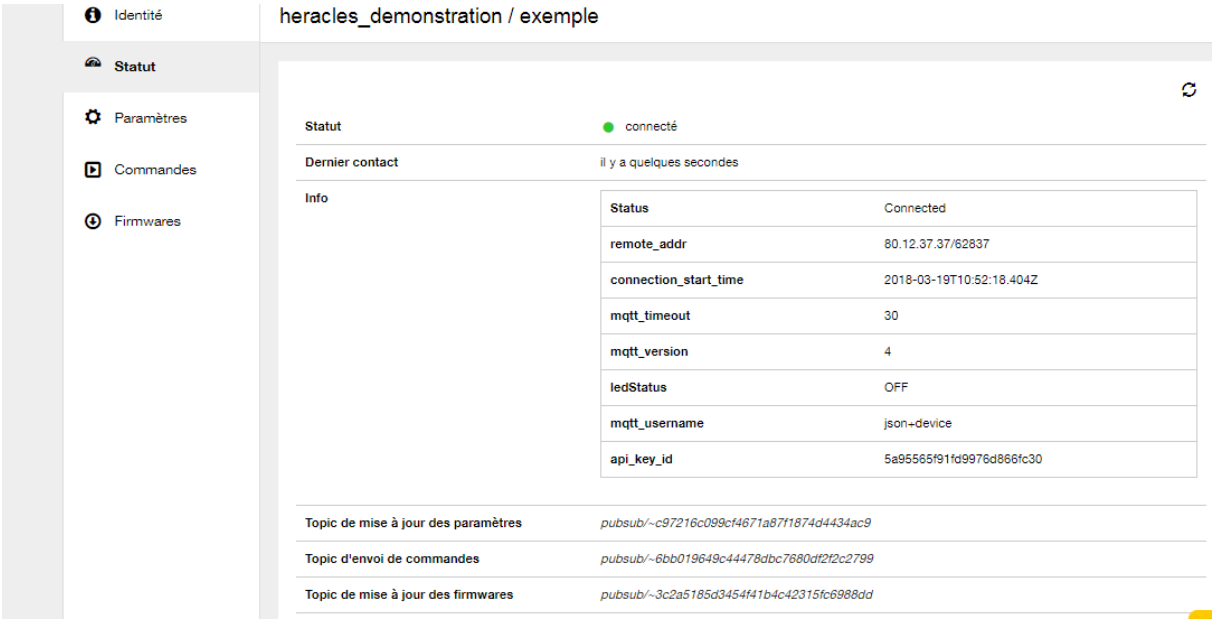
```
/***
* Function use to send the status of the device to the topic dev/info
*/
void sendStatus()
{
       //For more information on JsonObject library https://arduinojson.org/
       JsonObject& root = jsonBuffer.createObject();
       JsonObject & info = root.createNestedObject("info");
       info["ledStatus"] = LED_BUILTIN_STATUS ? "On" : "OFF";
       info["Status"] = mqtt.connected() ? "Connected" : "Not connected";
       publish(statusTopic, root, false, NULL);
}
```

It push data on the following topic :

```
const char* statusTopic = "dev/info";
```

3.4.2. Use of Live Objects portal to view/check the set of status

On the Datavenue Live Objects portal, the user can check the ‘status’ of its connected device:



3.5. Parameters

The device can declare one or many Live Objects "*parameters*" of device configurations.

Then, Live Objects can track the changes of the current value of device parameters, and allow users to set different target values for those parameters. Live Objects will then update the parameters on the device once it's connected and available.

3.5.1. Push parameters data

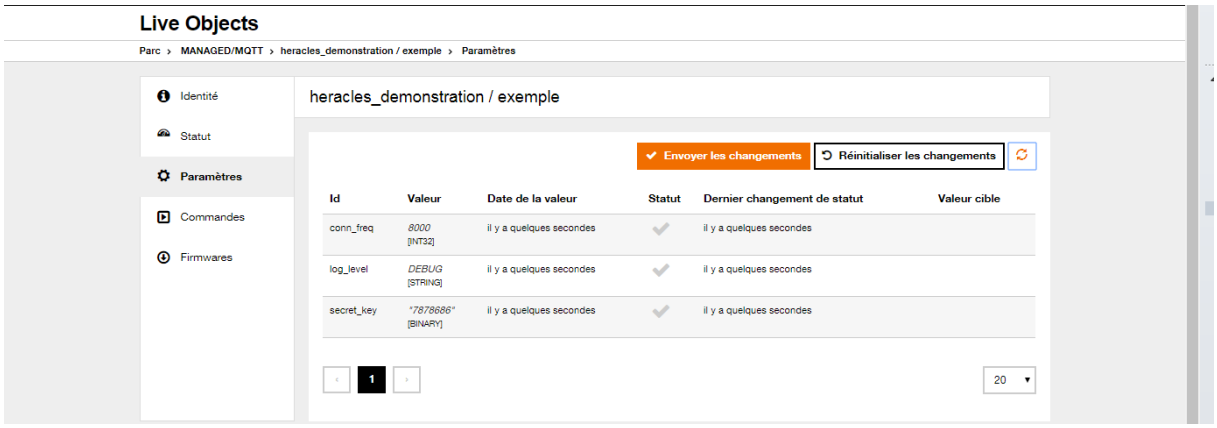
Parameters are send in the sendconfig() function to the following topic

```
const char* postCurrentCfgTopic = "dev/cfg";
```

```
void sendConfig()
{
	//For more information on JsonObject library https://arduinojson.org/
	JsonObject& root = jsonBuffer.createObject();
	JsonObject & cfg = root.createNestedObject("cfg");
	JsonObject & log_level = cfg.createNestedObject("log_level");
	log_level["t"] = "str";
	log_level["v"] = "DEBUG";
	JsonObject & secret_key = cfg.createNestedObject("secret_key");
	secret_key["t"] = "bin";
	secret_key["v"] = "Nzg3ODY4Ng==";
	JsonObject & conn_freq = cfg.createNestedObject("conn_freq");
	conn_freq["t"] = "i32";
	conn_freq["v"] = 8000;
	if (CID != "")
		root["cid"] = CID;
	publish(postCurrentCfgTopic, root, false, NULL);
}
```

3.5.2. Use of Live Objects Portal to set/change parameters

On the Datavenue Live Objects portal, the user can check the 'Parameters' of its connected device, but also change these initial values:



3.5.3. Receive parameters

To receive parameters we subscribe to the parameters configuration topic /dev/cfg/upd in the function subscribeToTopics()

```
void subscribeToTopics()
{
	// dev/cfg/upd
	mqtt.subscribe(receiveCfgTopic);

	// dev/cmd
	mqtt.subscribe(cmdTopic);

	// dev/rsc/upd
	mqtt.subscribe(updtRscTopic);
}
```

In the setup function we define a callback function

```
mqtt.setCallback(mqttCallback);
```

This callback is called when the device received data from one of the topic

3.6. Collected Data

The device can declare one or many Live Objects "*collected data*".

A collected data is defined by:

- streamId: identifier of the timeseries this message belongs to.
- Value: a set of user values (i.e.: temperature ...)
- Additional (and optional) information associated to this data stream:
  - model: a string identifying the schema used for the "value" part of the message, to avoid conflict at data indexing,
  - tags: list of strings associated to the message to convey extra-information.
- At each message published to the Live Objects platform, optional information

- **timestamp**: data/time associated with the message (using ISO 8601 format).
  If the timestamp is not specified, the data will be timestamped at the receipt by the Live Objects platform.
- **latitude, longitude**: details of the geo location associated with the message (in degrees).

3.6.1. Push collected data

You send data in the function sendData() to the following topic :

```
const char* posDataTopic = "dev/data"
```

```
void sendData()
{
	//For more information on JsonObject library https://arduinojson.org/

	JsonObject& root = jsonBuffer.createObject();
	String s = String(deviceid);
	s = String("!measuresTest");
	String m = "measuresTest_v1_3";
	root["s"] = s;
	root["m"] = m;
	JsonObject& v = root.createNestedObject("v");
	v["LedStatus"] = LED_BUILTIN_STATUS ? "1" : "0";
	v["Luminosity"] = lightValue;
	v["Temperature"] = temperatureValue;
	v["Humidity"] = humidityValue;
	v["Battery_Level"] = 100;
	v["Battery_Charging"] = 1;
	JsonArray& t = root.createNestedArray("t");
	//root["ts"] = "2017-09-19T10:00:00Z";
	t.add("Heracles");
	t.add("prototype");
	publish(posDataTopic, root, true, AlertSending);
}
```

3.6.2. Use of Live Objects Portal to view data stream

On the Datavenue Live Objects portal, the user can check the 'Collected Data' published by its connected device.

If you click on the icon  you will have see data in detailed


```
{
   "metadata": {
      "connector": "mqtt",
      "source": "urn:lo:nsid:heracles_demonstration:exemple"
   },
   "streamId": "!measuresTest",
   "created": "2018-03-19T11:47:22.998Z",
   "extra": null,
   "location": null,
   "model": "measuresTest_v1_3",
   "id": "5aafa34ac74a4969f28fd0c5",
   "value": {
      "Battery_Level": 100,
      "Temperature": 9,
      "LedStatus": "0",
      "Humidity": 31,
      "Luminosity": 22,
      "Battery_Charging": 1
   },
   "timestamp": "2018-03-19T11:47:22.995Z",
   "tags": [
      "Heracles",
      "prototype"
   ]
}
```


3.7. Commands

3.7.1. Subscribe to command

After the device subscribe from the topic, it can receive command from the web.

The will receive command in the mqtt callback function

```
/**
* Function call everytime the user receive a mqtt message
*/
void mqttCallback(char* topic, uint8_t * payload, unsigned int len) {
      …

       if (String(topic) == cmdTopic) // When receiving message from topic "dev/cmd"
       {
              Serial.println("cmd");
              JsonObject& root = jsonBuffer.parseObject(buffer);
              String tmp_C_CID = root["cid"];
              if (String(C_CID) != tmp_C_CID)
              {
                     AlertReceive();
                     C_CID = tmp_C_CID;
                     String req = root["req"];
                     if (req == SWITCH_OFF)
                     {
                            JsonObject& arg = root["arg"].asObject();;
                            int light = arg.get<int>("Light");
                            if (light == 0)
                                   LED_BUILTIN_STATUS = false;
```

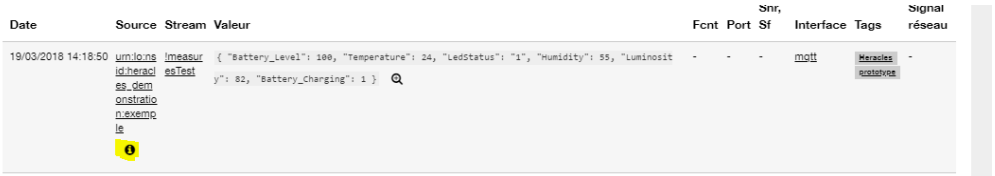
```
                        else
                                LED_BUILTIN_STATUS = true;
                        digitalWrite(LED_BUILTIN, LED_BUILTIN_STATUS);
                }
                sendReponseCommand();
        }
        else
        {
                Serial.println("CCID don't change");
        }
    }
    …
}
```

In the callback you can define the behavior of the application regarding the command called.

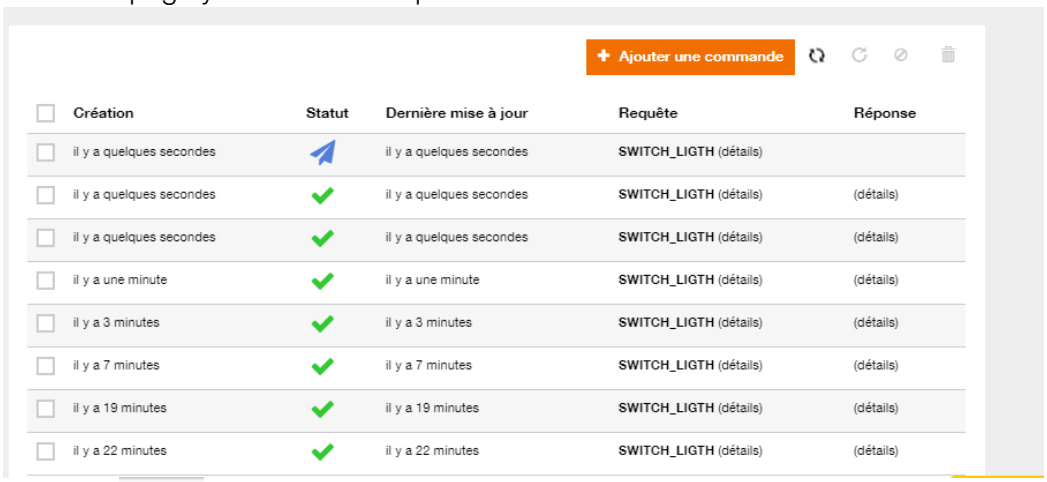
3.7.2. Use of Live Objects Portal to send a command

On the Live Objects Portal,

- Go to tab Data and click on the information device icon 



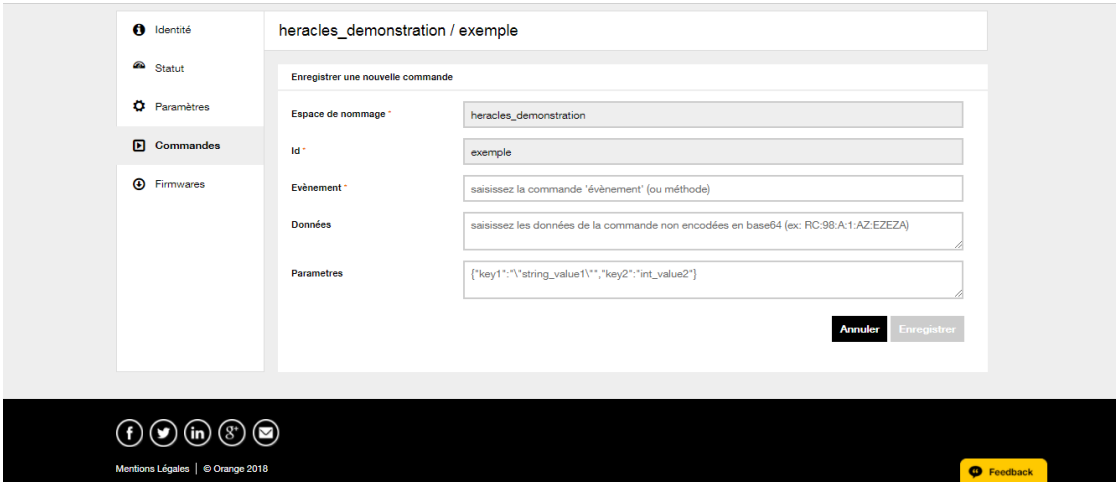
- Click on commands in the left menu
- From this page you can send mqtt command



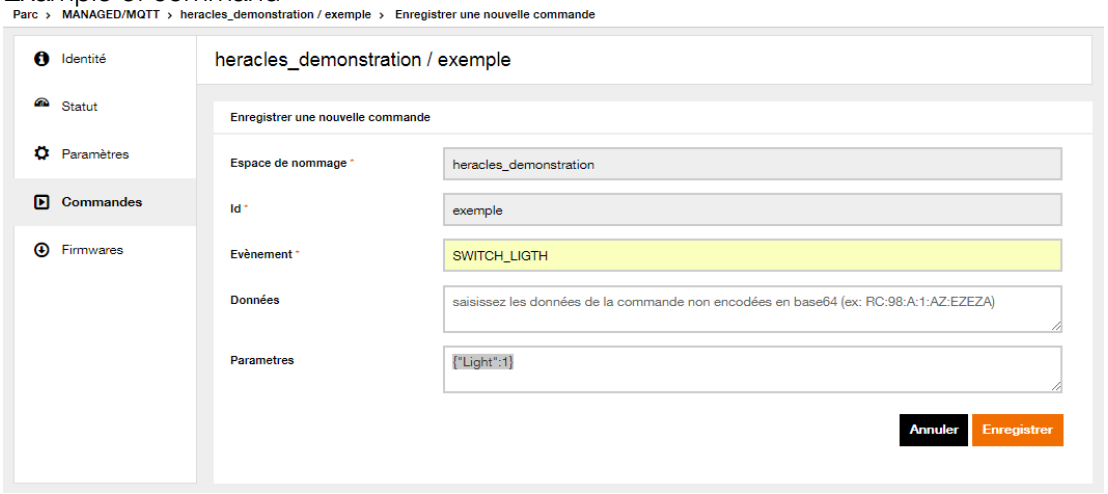
The icon  appears when the command is processed to the device

The icon  appears when the command has been received to the device
If you click on details you will see the details of the command or the details of the answer.

- Click on button '+ Add command"



Example of command



- Click on button 'Register'. And wait a few moment , refresh the web page

3.8. Firmware

3.8.1. Subscribe to the firmware topic

The device subscribe to the following topic to get new firmware :

- dev/rsc/upd

All the data coming from that topic will be managed on the mqtt callback function

```
void mqttCallback(char* topic, uint8_t * payload, unsigned int len) {


      …
      if (String(topic) == updtRscTopic)        // When receiving message from topic
dev/rsc/upd
      {
            Serial.println("update resource");
            JsonObject& root = jsonBuffer.parseObject(buffer);
```

```
            String tmp_R_CID = root["cid"];
            R_CID = tmp_R_CID;
            sendResources();
      }
}
```

The device will confirm to Live Object that it received the information about the firmware by sending back the stream Id of the firmware (R_CID).

```
// dev/rsc
void sendResources()
{
      JsonObject& root = jsonBuffer.createObject();
      JsonObject & rsc = root.createNestedObject("rsc");
      JsonObject & firmware = rsc.createNestedObject("firmware");
      firmware["v"] = "1_2";
      JsonObject & m = firmware.createNestedObject("m");
      m["username"] = "heracles";
      if (R_CID != "")
      {
            root["cid"] = R_CID;
            publish(updtResponseRscTopic, root, false, NULL);
      }
      else
      {
            publish(sendRscTopic, root, false, NULL);
      }
}
```

For more information about the update, click on the following link :
https://liveobjects.orange-business.com/doc/html/lo_manual.html#MQTT_DEV_RSC_UPD