IoT Soft Box Starter Kit

User Manual for iotsoftbox-mqtt library

Arduino Edition

Table of contents

1. Introduction

1.1. Document purpose

This document is a complete guide to IoT Soft Box SDK for Arduino presenting the following:

- Overview
- Getting started
- Detailed Features
- Additional Information

1.2. Reference documents

#	Origin	Title
1	Orange	Datavenue Live Objects - complete guide (1.14.15.)

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, so Orange worked on a new version of **the Live Objects iotsoftbox-mqtt library** compatible with Arduino platforms.

Also, as Arduino is open-source hardware, many manufacturers are building compatible variant of original Arduino designs. One good example is the Mediatek LinkIt-ONE. It uses a Mediatek MT2502A SoC which supports GSM/GPRS and Wi-Fi as communication connectivities. This card provides good connectivity and enough RAM for security functions.

2.3. IoT Soft Box

The Live Objects IoT Soft Box is a library to help developers make easy usage of Live Objects platform.

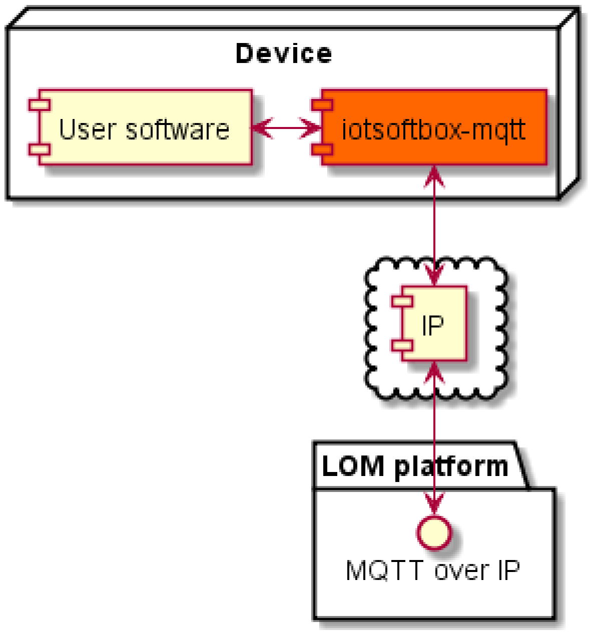


Figure 1 – IoT Soft Box integration in a system

The Live Objects platform is able to manage different formats (MQTT, HTTP, …) and several low level protocols (SMS, IP, …). The Live Objects IoT Soft Box is designed to work with MQTT over TCP w/wo TLS.

The IoT Soft Box can run on devices connected to Internet through Ethernet, Wifi, GPRS or any other IP connection. It doesn't embed SSL/TLS capability. The connection layer (Wifi library, GPRS shield, Live Booster Heracles modem, …) should manage the SSL/TLS layer in order to enable data transfer security.

The library (iotsoftbox-mqtt) is linked to the following third-party existing libraries:

- Embedded MQTT C/C++ Client Libraries (eclipse paho). This library is available here.
- JSMN, a simple C library only used to parse the received JSON messages. The JSMN is available here.

3. Getting started

3.1. Hardware Environment and compatibility

As of today, we are compatible with the following boards:

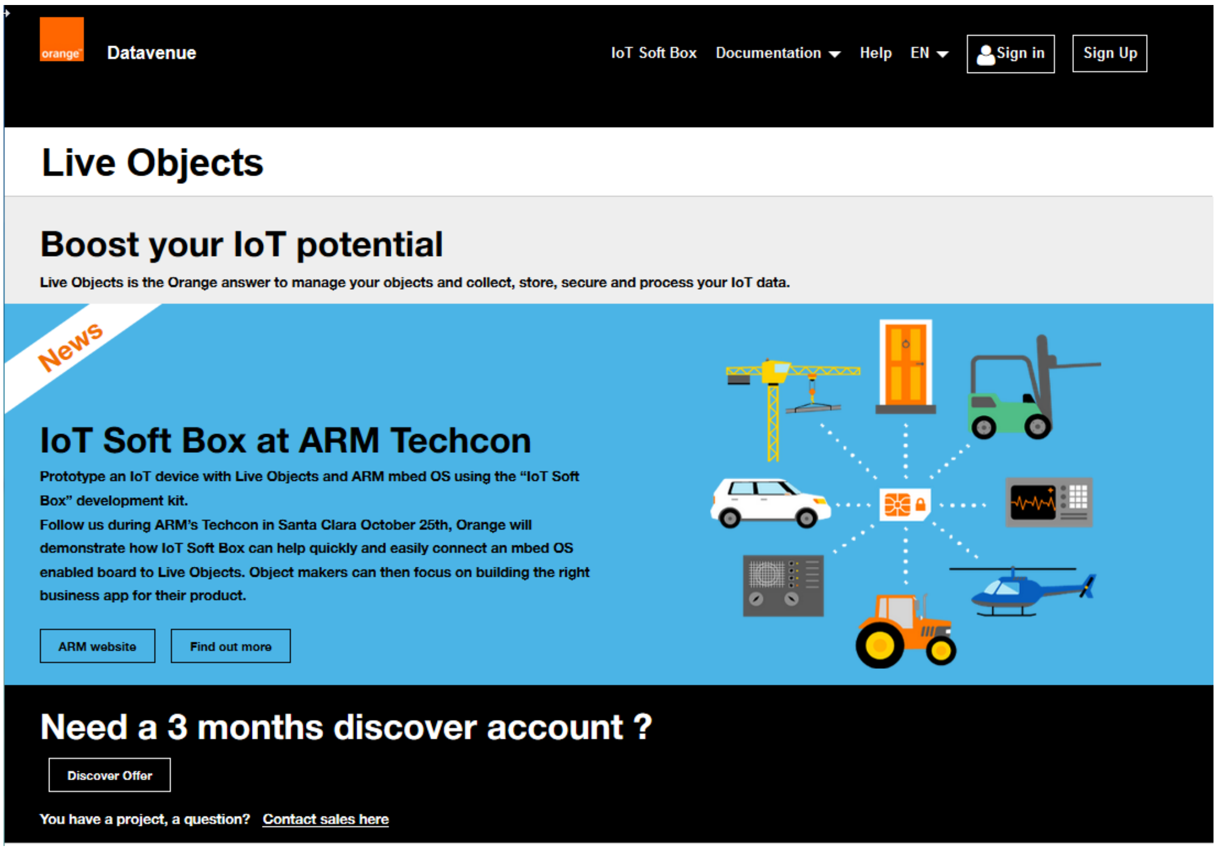
- Mediatek LinkIt One
- Arduino MEGA ADK
- Other Arduino boards should be compatible but were not tested.

3.2. Access to Live Objects

3.2.1. Account creation

In order to use Live Objects, you need to have a dedicated account on the service.

1. Go to Live Objects portal (https://liveobjects.orange-business.com/).



Live Objects

Boost your IoT potential

Live Objects is the Orange answer to manage your objects and collect, store, secure and process your IoT data.

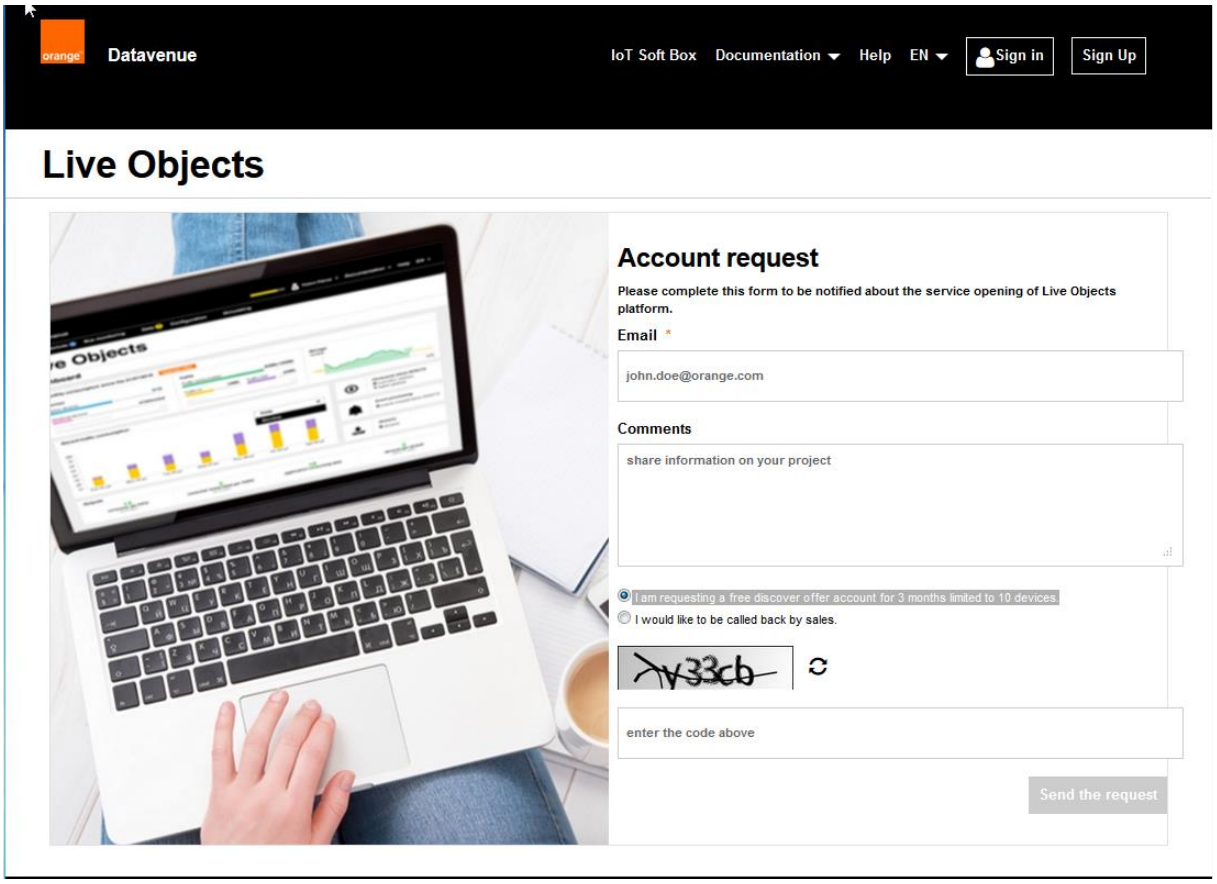
IoT Soft Box at ARM Techcon

Prototype an IoT device with Live Objects and ARM mbed OS using the “IoT Soft Box” development kit.
Follow us during ARM's Techcon in Santa Clara October 25th, Orange will demonstrate how IoT Soft Box can help quickly and easily connect an mbed OS enabled board to Live Objects. Object makers can then focus on building the right business app for their product.

Need a 3 months discover account ?

You have a project, a question? Contact sales here

2. Click on '**Discover Offer**' button (or Sign Up) and fill the form, checking option 'I am requesting a free discover offer account for 3 months limited to 10 devices'.



Live Objects

Account request

Please complete this form to be notified about the service opening of Live Objects platform.

Email *

john.doe@orange.com

Comments

share information on your project

[✓ I am requesting a free discover offer account for 3 months limited to 10 devices.]

I would like to be called back by sales.

enter the code above

Send the request

3. Then you will receive an e-mail to activate your Live Objects account.

Hello,

You just subscribed to Live Objects portal.

Please, use this link to activate your account 'john' and define your password.

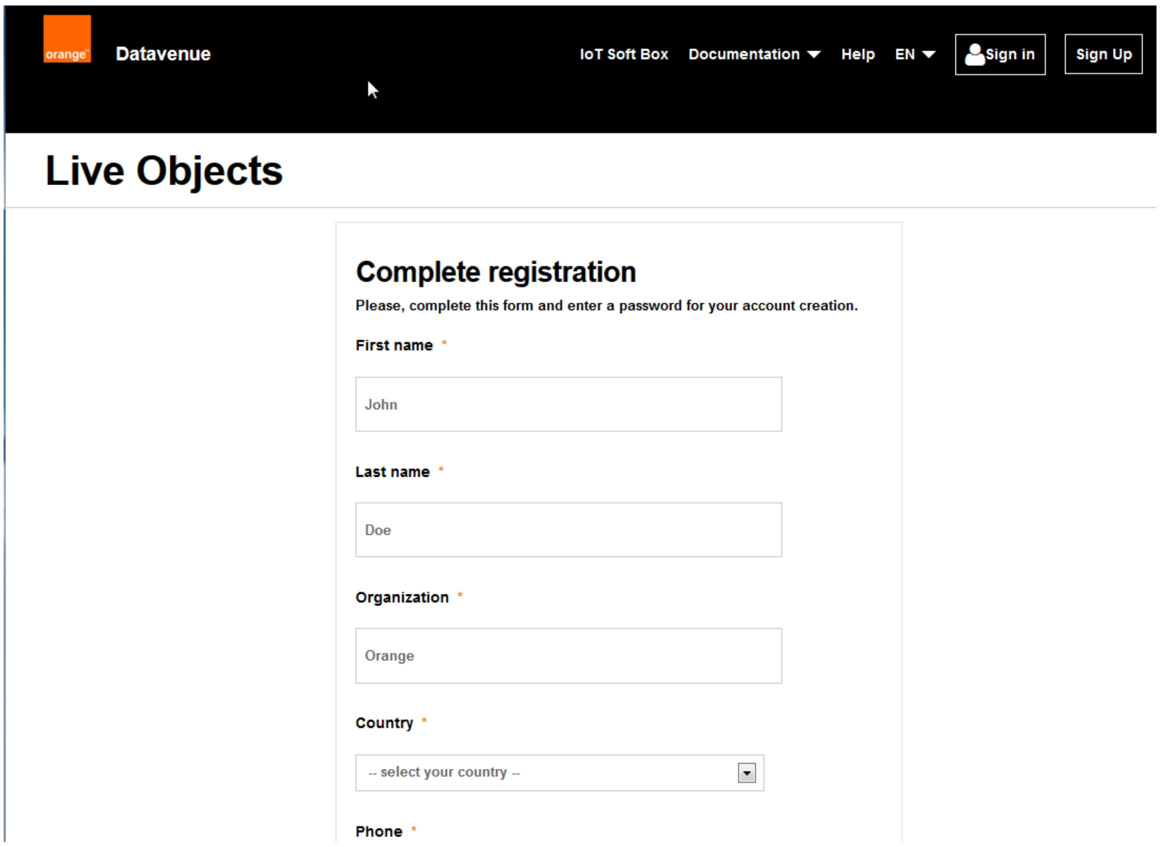
'john' Account activation

Thank you for your trust.

Orange Business Service Customer Support.

This is an automatically generated email, please do not reply.

4. Follow the link, fill the form, and click on ‘Validate’.



Live Objects

Complete registration

Please, complete this form and enter a password for your account creation.

First name *

John

Last name *

Doe

Organization *

Orange

Country *

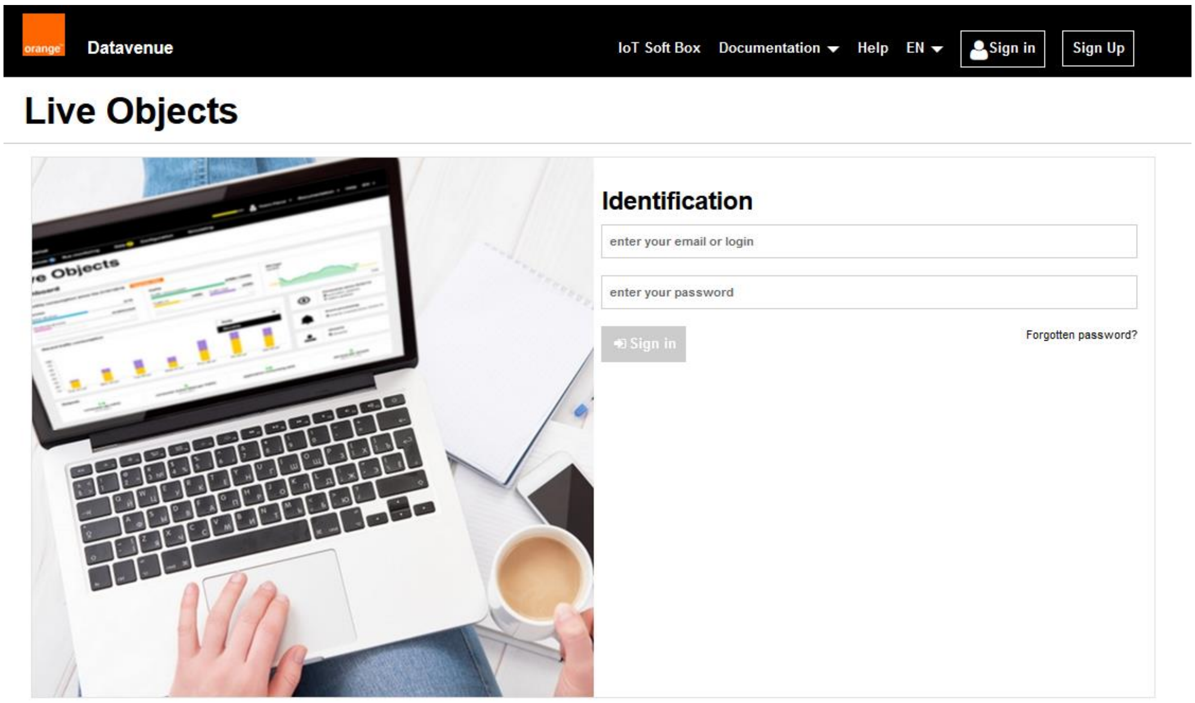
-- select your country --

Phone *

5. Now, you can go back to Datavenue Live Objects portal and sign in. Once logged, select the ‘configuration’ tab to create a new API key.

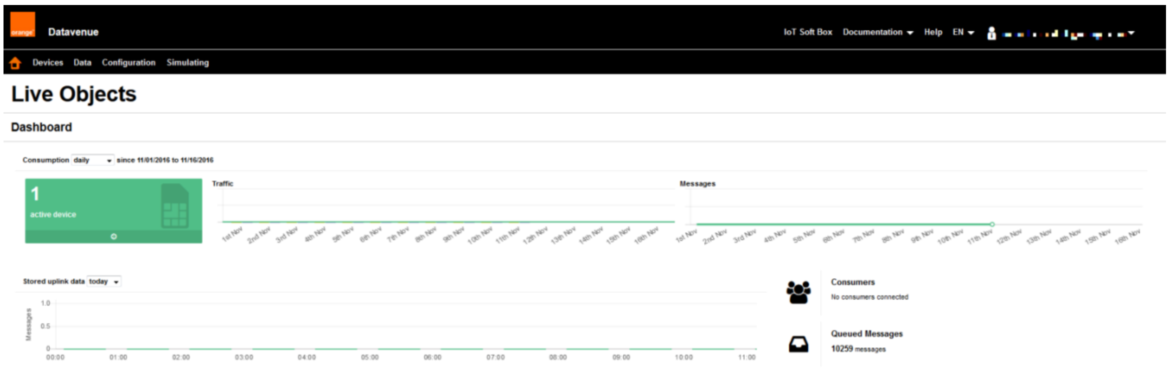
3.2.2. Log in

To log in to Live Objects web portal, connect to liveobjects.orange-business.com using your web-browser:



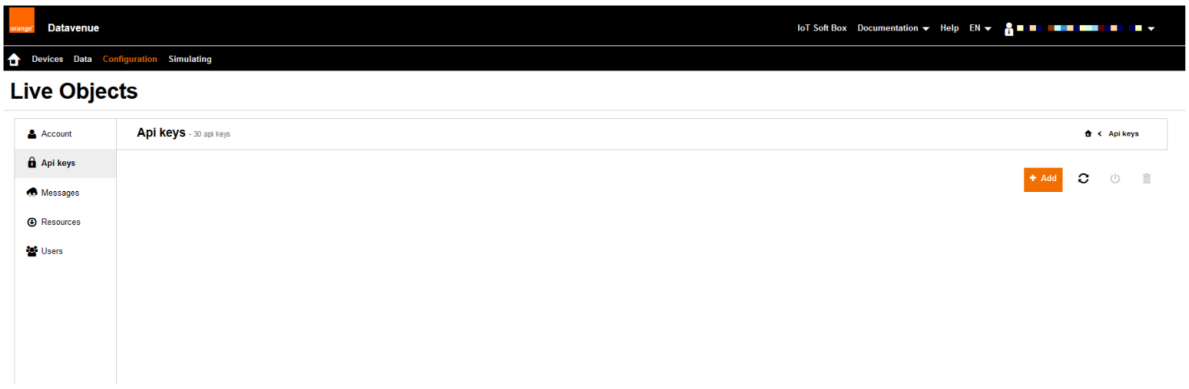
1. Fill the “Log in” form with your credentials:
    - your email address,
    - the password set during the activation phase,
2. Then click on the “Log in” button.

If the credentials are correct, a success message is displayed and you are redirected to your “home” page:

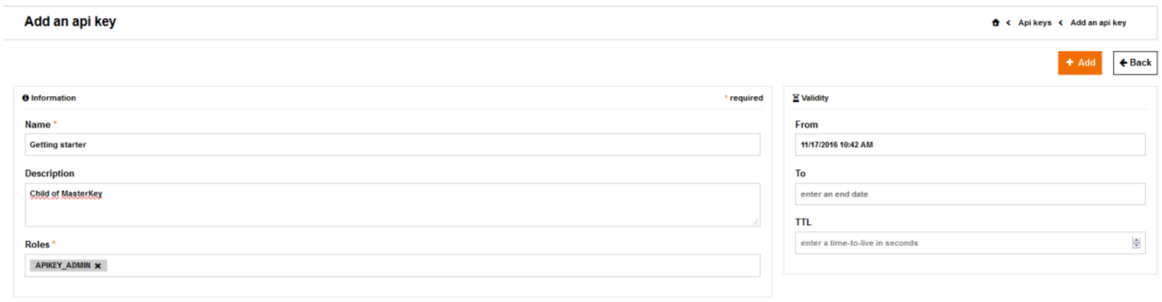


3.2.3. API Key creation

To get a device or an application communicating with Live Objects, you will need to create an API Key in the “Configuration” menu. On the left menu, click on "Api keys" and create a new API key. This key will be necessary to set up a connection with the public interfaces (MQTT and REST) of Live Objects.



As a security measure, you cannot retrieve the API Key again after closing the API key creation results page. So, note it down to work with the MQTT client, during the scope of this getting started.


Add an api key

Api keys < Add an api key

+ Add ← Back

Information * required

Name *
Getting starter

Description
Child of MasterKey

Roles *
APIKEY_ADMIN

Validity

From
11/17/2016 10:42 AM

To
enter an end date

TTL
enter a time-to-live in seconds

3.3. Live Objects IoT examples using iotsoftbox-mqtt library

3.3.1. 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
    - The current Configuration Parameters
    - The current Resources
- Subscribes to Live Objects topics to receive notifications
    - Configuration Parameters update request
    - Resource update request
    - Command request
- then the application waits for an event:
    - From Live Objects platform to:
        - Update "Configuration Parameters"
        - Update one "Resource": message or image
        - Process a "Command": RESET or LED
    - From application simulating some data publish operations.
    - And if the connection is lost, restart at the first step trying to connect again to the Live Objects platform.

Three different examples **with the same functionalities** are included in the library :

Example	Target	Connectivity
liveobjects_sample_basic_mdk	Mediatek LinkIt ONE	Internal LinkIt ONE GSM module
liveobjects_sample_Heracles_basic_avr	Generic Arduino board	**LiveBooster Heracles modem**
liveobjects_sample_Heracles_basic_mdk	Mediatek LinkIt ONE	**LiveBooster Heracles modem**

To use an external LiveBooster Heracles modem, you need to install the **LiveBooster-Heracles-Arduino** library available here : https://github.com/Orange-OpenSource/LiveBooster-Heracles-Arduino

3.3.2. Packages dependencies

The example applications have been built and tested with the following packages:

1) *Paho MQTT embedded-c (MQTTPacket and MQTTClient)*
   https://github.com/eclipse/paho.mqtt.embedded-c/#02323e1093f0414c1adcf03559e03a890b5f3a84

2) *jsmn*
https://github.com/zserge/jsmn/#1682c32e9ae5990ddd0f0e907270a0f6dde5cbe9

Note that these packages are included in the ZIP file of the iotsoftbox-mqtt library .

3.3.3. Configure workstation

Using the Arduino IDE allows to abstract the host platform.

This example is given for a generic Arduino device as a target. Step 2 is only if you're using a Mediatek LinkIt^{tm} ONE as a target.

1. Install the Arduino IDE (tested version: 1.8.5)

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

2. Launch Arduino IDE
3. Install additional packages/libraries (to use your boards and shields)

- If you're using the LinkIt One board, you will need to download a SDK

```
Get it from : https://docs.labs.mediatek.com/resource/linkit-
one/en/getting-started/get-started-on-windows/install-the-arduino-
ide-and-linkit-one-sdk
```

4. Install the iotsoftbox-mqtt library

To install the library follow the procedure :

- Download the ZIP file from https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-arduino
- Open your Arduino IDE and add this library through the sketch menu : **Sketch -> Include Library** -> **Add .ZIP Library** and select the ZIP file (iotsoftbox_mqtt_arduino.zip)
- Update the iotsoftbox library configuration files located in the directory `<Arduino user dir>\libraries\iotsoftbox_mqtt_arduino\src\config`
  - *Liveobjects_dev_config.h* , optionally to change some parameters to tune the iotsoftbox library.

- Open example sketch **File** -> **Examples** -> **LiveObjects iotsoftbox library** -> …
- **You need to update the .ino sketch file to set your Live Objects Tenant API key.**
- For LinkIt One board with the internal modem, you may need to edit `liveobjects_sample_basic_mdk.h` file to set your SIM parameters: **GPRS_APN** , **GPRS_USERNAME** , **GPRS_PASSWORD**

- Don't forget to select the correct board
  - Mediatek LinkIt One:

        - Tools -> Boards -> LinkIt One
        - Tools -> Programmer -> LinkIt Firmware Updater
    - Arduino MEGA ADK:
        - Tools -> Boards -> Arduino Mega ADK

3.3.4. Build

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

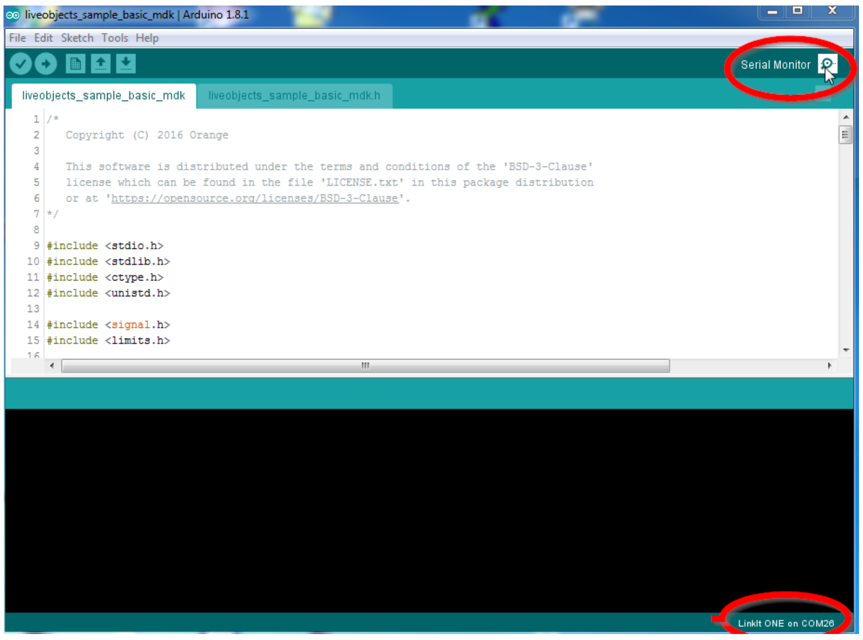
3.3.5. Launch

3.3.5.1. Upload the sample on your Arduino (or Mediatek) through the 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**.

3.3.5.2. Excute the application

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


liveobjects_sample_basic_mdk | Arduino 1.8.1
File Edit Sketch Tools Help
Serial Monitor
liveobjects_sample_basic_mdk
liveobjects_sample_basic_mdk.h
LinkIt ONE on COM26

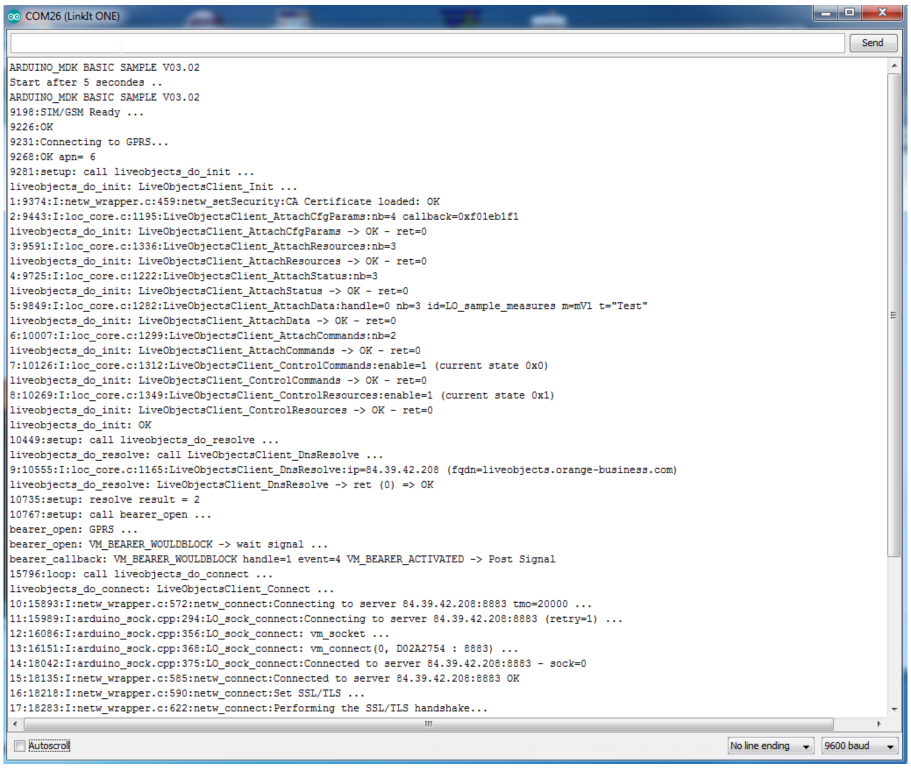
```
/*
   Copyright (C) 2016 Orange

   This software is distributed under the terms and conditions of the 'BSD-3-Clause'
   license which can be found in the file 'LICENSE.txt' in this package distribution
   or at 'https://opensource.org/licenses/BSD-3-Clause'.
*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>

#include <signal.h>
#include <limits.h>
```

The following window is open (with Mediatek LinkIt One board):



```
ARDUINO_MDK BASIC SAMPLE V03.02
Start after 5 secondes ..
ARDUINO_MDK BASIC SAMPLE V03.02
9198:SIM/GSM Ready ...
9226:OK
9231:Connecting to GPRS...
9268:OK apn= 6
9281:setup: call liveobjects_do_init ...
liveobjects_do_init: LiveObjectsClient_Init ...
1:9374:I:netw_wrapper.c:459:netw_setSecurity:CA Certificate loaded: OK
2:9443:I:loc_core.c:1195:LiveObjectsClient_AttachCfgParams:nb=4 callback=0xf01eb1f1
liveobjects_do_init: LiveObjectsClient_AttachCfgParams -> OK - ret=0
3:9591:I:loc_core.c:1336:LiveObjectsClient_AttachResources:nb=3
liveobjects_do_init: LiveObjectsClient_AttachResources -> OK - ret=0
4:9725:I:loc_core.c:1222:LiveObjectsClient_AttachStatus:nb=3
liveobjects_do_init: LiveObjectsClient_AttachStatus -> OK - ret=0
5:9849:I:loc_core.c:1282:LiveObjectsClient_AttachData:handle=0 nb=3 id=LO_sample_measures m=mV1 t="Test"
liveobjects_do_init: LiveObjectsClient_AttachData -> OK - ret=0
6:10007:I:loc_core.c:1299:LiveObjectsClient_AttachCommands:nb=2
liveobjects_do_init: LiveObjectsClient_AttachCommands -> OK - ret=0
7:10126:I:loc_core.c:1312:LiveObjectsClient_ControlCommands:enable=1 (current state 0x0)
liveobjects_do_init: LiveObjectsClient_ControlCommands -> OK - ret=0
8:10269:I:loc_core.c:1349:LiveObjectsClient_ControlResources:enable=1 (current state 0x1)
liveobjects_do_init: LiveObjectsClient_ControlResources -> OK - ret=0
liveobjects_do_init: OK
10449:setup: call liveobjects_do_resolve ...
liveobjects_do_resolve: call LiveObjectsClient_DnsResolve ...
9:10555:I:loc_core.c:1165:LiveObjectsClient_DnsResolve:ip=84.39.42.208 (fqdn=liveobjects.orange-business.com)
liveobjects_do_resolve: LiveObjectsClient_DnsResolve -> ret (0) => OK
10735:setup: resolve result = 2
10767:setup: call bearer_open ...
bearer_open: GPRS ...
bearer_open: VM_BEARER_WOULDBLOCK -> wait signal ...
bearer_callback: VM_BEARER_WOULDBLOCK handle=1 event=4 VM_BEARER_ACTIVATED -> Post Signal
15796:loop: call liveobjects_do_connect ...
liveobjects_do_connect: LiveObjectsClient_Connect ...
10:15893:I:netw_wrapper.c:572:netw_connect:Connecting to server 84.39.42.208:8883 tmo=20000 ...
11:15989:I:arduino_sock.cpp:294:LO_sock_connect:Connecting to server 84.39.42.208:8883 (retry=1) ...
12:16086:I:arduino_sock.cpp:356:LO_sock_connect: vm_socket ...
13:16151:I:arduino_sock.cpp:368:LO_sock_connect: vm_connect(0, D02A2754 : 8883) ...
14:18042:I:arduino_sock.cpp:375:LO_sock_connect:Connected to server 84.39.42.208:8883 - sock=0
15:18135:I:netw_wrapper.c:585:netw_connect:Connected to server 84.39.42.208:8883 OK
16:18218:I:netw_wrapper.c:590:netw_connect:Set SSL/TLS ...
17:18283:I:netw_wrapper.c:622:netw_connect:Performing the SSL/TLS handshake...
```

3.3.5.3. 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.

4. Detailed Features

4.1. General

The Live Objects IoT Soft Box is a library providing features to connect embedded device to the Datavenue Live Objects platform.

Today, a library dedicated to Arduino boards is available here, library called **LiveObjects-iotSoftbox-mqtt-arduino.**

The **LiveObjects-iotSoftbox-mqtt** library provides APIs to help developers create their embedded IoT applications. The API is written in C.

The **LiveObjects-iotSoftbox-mqtt** library uses Live Objects 'Device' mode: a single MQTT connection is associated with the device, and JSON messages can be exchanged to support various *Device Management* and *Data* features. See "Device" mode paragraph in Live Objects User Manual to have a full description.

IoT Soft Box available features are:

- Connection to the user tenant of Live Objects platform w/wo security (TLS)
- Device Management
- Status
- Configuration Parameters
- Collected data
- Commands
- Resources

4.2. Connectivity

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.

Therefore the user has only to define:

- Tenant *ApiKey* parameter, in the main file (typically the .ino Arduino sketch file) :

```
/** Here, set your LiveObjects API key. It is mandatory to run the application.
 *
 * C_LOC_CLIENT_DEV_API_KEY_P1 must be the first sixteen char of the ApiKey
 * C_LOC_CLIENT_DEV_API_KEY_P1 must be the last sixteen char of the ApiKey
 *
 * If your APIKEY is 0123456789abcdeffedcba9876543210 then
 * it should look like this :
 *
 * #define C_LOC_CLIENT_DEV_API_KEY_P1      0x0123456789abcdef
 * #define C_LOC_CLIENT_DEV_API_KEY_P2      0xfedcba9876543210
 *
 * */
#define C_LOC_CLIENT_DEV_API_KEY_P1             0x0123456789abcdef
#define C_LOC_CLIENT_DEV_API_KEY_P2             0xfedcba9876543210
```

- if the TLS is enabled or not, in liveobjects_dev_param.h:

To enable or disable security :

You just have to set the SECURITY_ENABLED flag to 1 (enable) or 0 (disable) in liveobjects_dev_param.h

Also if necessary, the endpoint parameters can be overwritten by parameters defined in this user header file: liveobjects_dev_params.h.

```
/* Only used to overwrite the LiveOjects Server settings :*/
/* IP address, TCP port, Connection timeout in milliseconds.*/
#define LOC_SERV_IP_ADDRESS                  "XXXX"
#define LOC_SERV_PORT                        XXXX
#define LOC_SERV_TIMEOUT                     XXXX
```

4.3. Device

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 # / !.

These two parameters are specified in the main file (typically the .ino Arduino sketch file) :

```
// Default LiveObjects device settings : name space and device identifier
#define LOC_CLIENT_DEV_NAME_SPACE            "LiveObjectsSample"
#define LOC_CLIENT_DEV_ID                    "LO_arduino_dev01"
```

4.4. Thread Models: Multi-thread or single thread.

The library offers both thread models to build the user embedded application:

1. Single thread. The user application has to schedule all tasks (or to call functions) in one same thread.
2. Multi-thread: A function of iotsoftbox-mqtt library allows the creation/activation of specific thread:
    - To maintain the TCP connection (w/wo TLS) to the Live Objects platform
    - To process all events from/to the Live Objects platform.

Note: that our samples running with Arduino uses the single-thread model. The multi-thread model is available in our ARM mbed and Linux libraries.

4.5. Status

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

4.5.1. Attach a set of ‘status’ data

At any moment, the application can attach one or many set (or group) of ‘status’ data by calling the function:

```
int LiveObjectsClient_AttachStatus(const LiveObjectsD_Data_t* status_ptr,
              int32_t status_nb);
```

In the sample application:

```
appv_hdl_status = LiveObjectsClient_AttachStatus(appv_set_status, SET_STATUS_NB);
```

The set of ‘status’ data is defined by an array of `LiveObjectsD_Data_t` elements. For example:

```
#define APPV_VERSION "ARDUINO_MDK BASIC SAMPLE V03.02"
int32_t appv_status_counter = 0;
char appv_status_message[150] = "READY";

/// Set of status
LiveObjectsD_Data_t appv_set_status[] = {
              { LOD_TYPE_STRING_C, "sample_version", APPV_VERSION, 1 },
              { LOD_TYPE_INT32, "sample_counter", &appv_status_counter, 1 },
              { LOD_TYPE_STRING_C, "sample_message", appv_status_message, 1 }
};
```

4.5.2. Push a set of 'status' data

When 'status' data change, the application must call the `LiveObjectsClient_PushStatus( )` function to notify the Datavenue Live Objects platform (publishing a MQTT message on the dev/info topic):

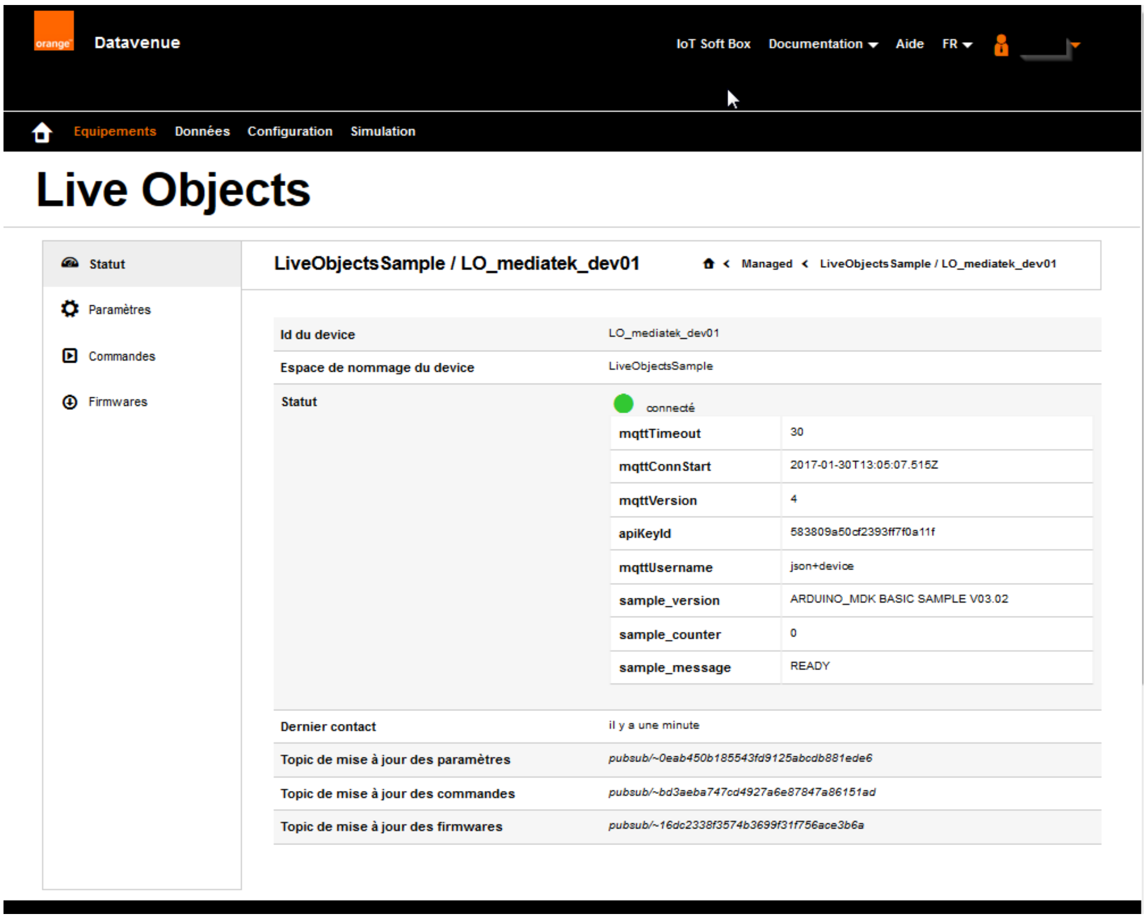
```
ret = LiveObjectsClient_PushStatus(appv_hdl_status);
```

Note:

- if the status data is attached before connecting to the platform, the 'status' data will be automatically pushed as soon as the MQTT connection is established with the Live Objects platform.

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

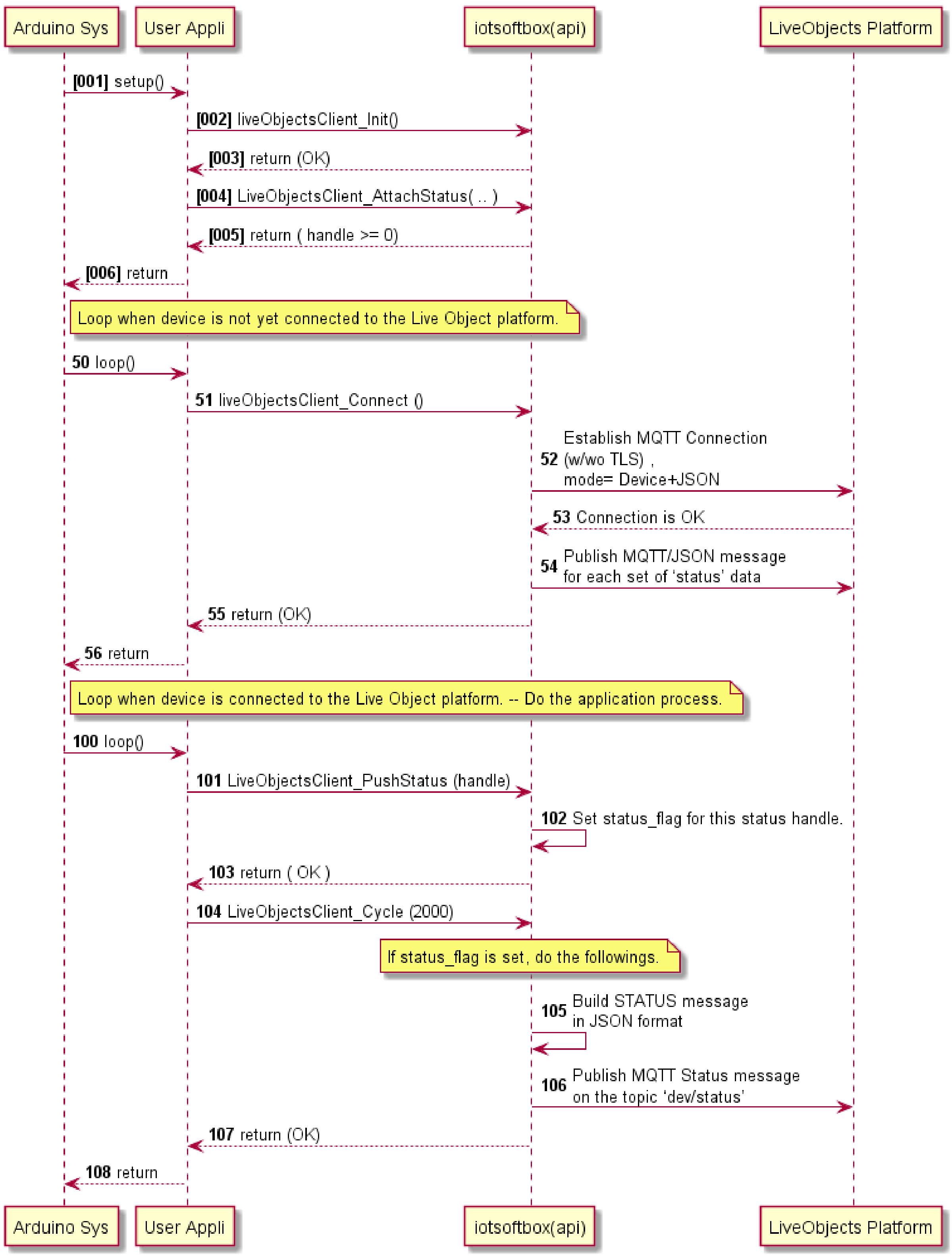
On the Datavenue Live Objects portal, the user can check the 'status' of its connected device:



LiveObjectsSample / LO_mediatek_dev01

Id du device	LO_mediatek_dev01
Espace de nommage du device	LiveObjectsSample
Statut	● connecté
	mqttTimeout: 30
	mqttConnStart: 2017-01-30T13:05:07.515Z
	mqttVersion: 4
	apiKeyId: 583809a50cf2393ff7f0a11f
	mqttUsername: json+device
	sample_version: ARDUINO_MDK BASIC SAMPLE V03.02
	sample_counter: 0
	sample_message: READY
Dernier contact	il y a une minute
Topic de mise à jour des paramètres	*pubsub/~0eab450b185543fd9125abcdb881ede6*
Topic de mise à jour des commandes	*pubsub/~bd3aeba747cd4927a6e87847a86151ad*
Topic de mise à jour des firmwares	*pubsub/~16dc2338f3574b3699f31f756ace3b6a*

4.5.4. Sequence Diagram


Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform
[001] setup()
[002] liveObjectsClient_Init()
[003] return (OK)
[004] LiveObjectsClient_AttachStatus(..)
[005] return (handle >= 0)
[006] return
Loop when device is not yet connected to the Live Object platform.
50 loop()
51 liveObjectsClient_Connect ()
52 Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
53 Connection is OK
54 Publish MQTT/JSON message for each set of 'status' data
55 return (OK)
56 return
Loop when device is connected to the Live Object platform. -- Do the application process.
100 loop()
101 LiveObjectsClient_PushStatus (handle)
102 Set status_flag for this status handle.
103 return (OK)
104 LiveObjectsClient_Cycle (2000)
If status_flag is set, do the followings.
105 Build STATUS message in JSON format
106 Publish MQTT Status message on the topic 'dev/status'
107 return (OK)
108 return
Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform

4.6. 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.

4.6.1. Attach a set of configuration parameters

Application can declare/attach only one set of configuration parameters to the iotsoftbox-mqtt library by calling the function:

```
int LiveObjectsClient_AttachCfgParams(
              const LiveObjectsD_Param_t *param_ptr,
              int32_t param_nb,
              LiveObjectsD_CallbackParams_t callback);
```

In the sample application:

```
ret = LiveObjectsClient_AttachCfgParams(appv_set_param, SET_PARAM_NB,
       main_cb_param_udp);
```

Where:

1. The set of 'parameters' data is defined by an array of `LiveObjectsD_Param_t` elements.
   In the sample application:

```
// definition of identifer for each kind of parameters
#define PARM_IDX_NAME 1
#define PARM_IDX_TIMEOUT 2
#define PARM_IDX_THRESHOLD 3
#define PARM_IDX_GAIN 4

/// Set of configuration parameters
LiveObjectsD_Param_t appv_set_param[] = {
       { PARM_IDX_NAME, { LOD_TYPE_STRING_C, "name", appv_conf.name, 1 } },
       { PARM_IDX_TIMEOUT, { LOD_TYPE_UINT32, "timeout", (void*) &appv_cfg_timeout, 1  } },
       { PARM_IDX_THRESHOLD, { LOD_TYPE_INT32,    "threshold", &appv_conf.threshold, 1 } },
       { PARM_IDX_GAIN, { LOD_TYPE_FLOAT, "gain", &appv_conf.gain, 1   } }
};
#define SET_PARAM_NB (sizeof(appv_set_param) / sizeof(LiveObjectsD_Param_t))
```

And the configuration parameters are defined and initialized as:

```
// a structure containing various kind of parameters (char[], int and float)
struct conf_s {
	char name[20];
	int32_t threshold;
	float gain;
} appv_conf = { "TICTAC", -3, 1.05 };
```

2. The application specifies the callback function (i.e. `paramUdpdateCb`) which will be called when a request is received from the Live Objects platform to change the value of parameter.

```
int main_cb_param_udp(const LiveObjectsD_Param_t *param_ptr, const void *value,
	int len) {
	if (param_ptr == NULL) {
		return -1;
	}
	switch (param_ptr->parm_uref) {
	case PARM_IDX_NAME: {
		…
		if (paramIsOk) {
			return 0;
		}
		break;
	}
	case PARM_IDX_TIMEOUT: {
		…
		if (paramIsOk) {
			return 0;
		}
		break;
	case PARM_IDX_THRESHOLD: {
		…
		if (paramIsOk) {
			return 0;
		}
		break;
	case PARM_IDX_GAIN: {
		…
		if (paramIsOk) {
			return 0;
		}
		break;
	}
	return -1;
}
```

With the Switch statement you can adapt the behavior of your app for each param.

Notes:

- When the user callback returns 0 to accept the new value for a 'primitive' parameter (integer, float ...), the iotsoftbox-mqtt library updates the value of this configuration parameter. But for the 'c-string' parameter, the user application has to copy the value in the good memory place (with the good size).
- The 'parameters' data will be automatically pushed as soon as the MQTT connection is established with the LiveObjects platform.

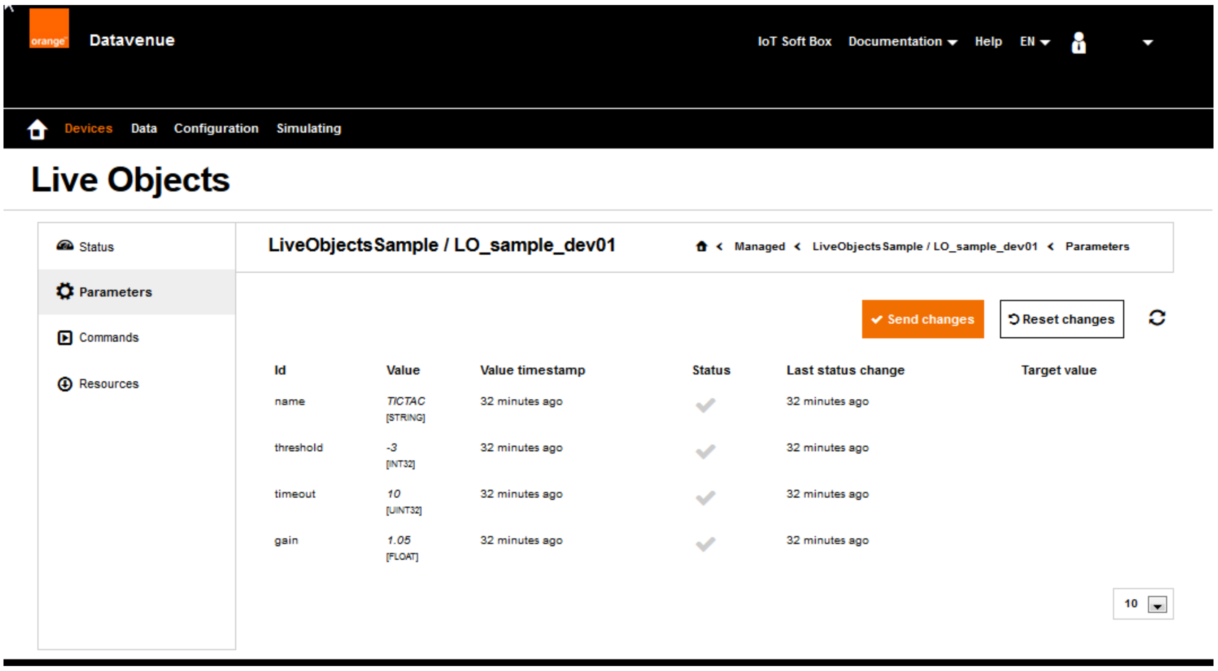
4.6.2. Push a set of configuration parameters

The application can call the `LiveObjectsClient_PushCfgParams( )` function to notify the Datavenue Live Objects platform (publishing a MQTT message on the dev/cfg topic) that the current configuration is updated:

```
int LiveObjectsClient_PushCfgParams(void);
```

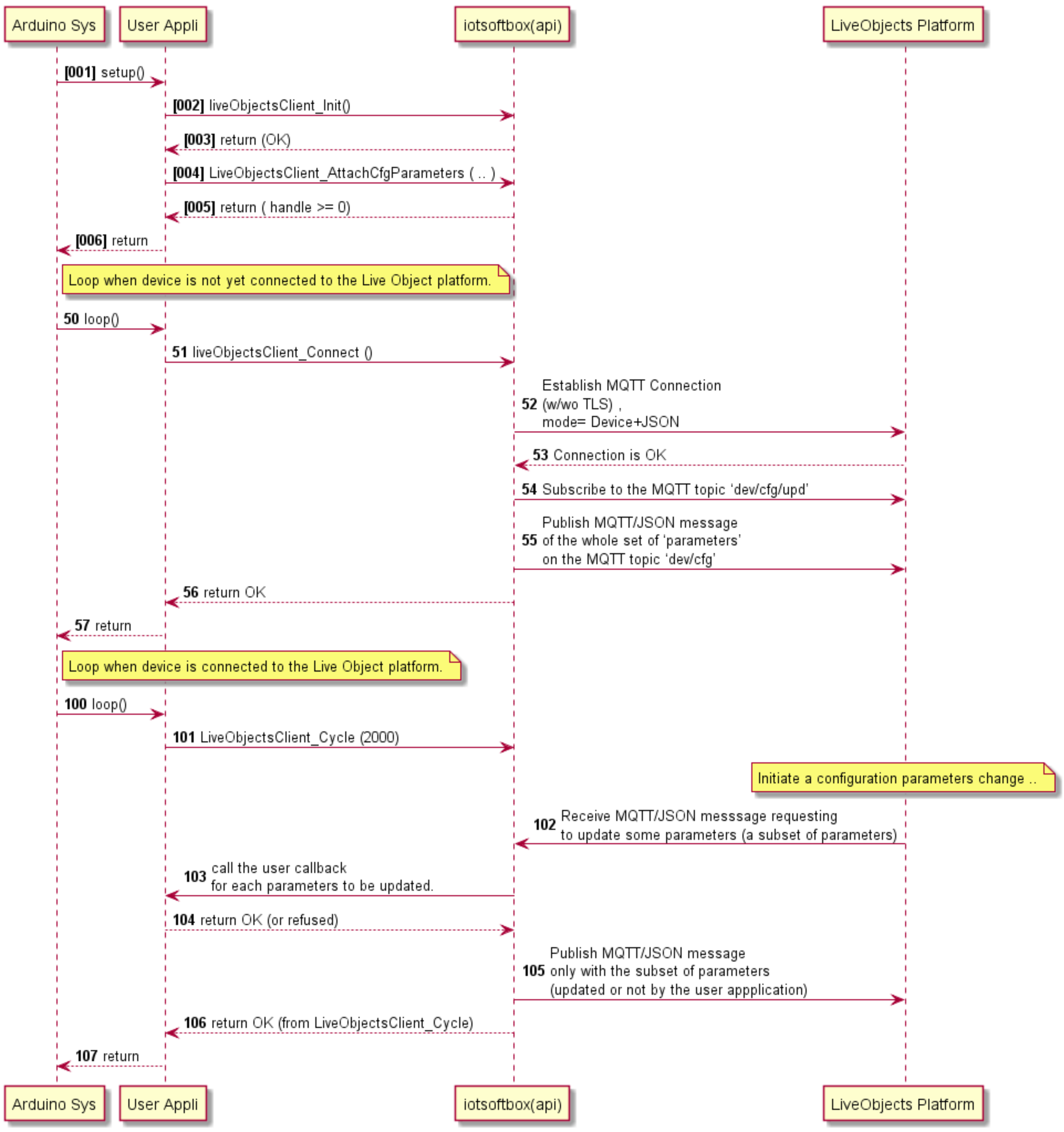
4.6.3. Use of Live Objects Portal to set/change parameters

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



Id	Value	Value timestamp	Status	Last status change	Target value
name	*TICTAC* [STRING]	32 minutes ago	✓	32 minutes ago	
threshold	*-3* [INT32]	32 minutes ago	✓	32 minutes ago	
timeout	*10* [UINT32]	32 minutes ago	✓	32 minutes ago	
gain	*1.05* [FLOAT]	32 minutes ago	✓	32 minutes ago	

4.6.4. Sequence Diagram


Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform
[001] setup()
[002] liveObjectsClient_Init()
[003] return (OK)
[004] LiveObjectsClient_AttachCfgParameters (..)
[005] return (handle >= 0)
[006] return
Loop when device is not yet connected to the Live Object platform.
50 loop()
51 liveObjectsClient_Connect ()
52 Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
53 Connection is OK
54 Subscribe to the MQTT topic 'dev/cfg/upd'
55 Publish MQTT/JSON message of the whole set of 'parameters' on the MQTT topic 'dev/cfg'
56 return OK
57 return
Loop when device is connected to the Live Object platform.
100 loop()
101 LiveObjectsClient_Cycle (2000)
Initiate a configuration parameters change ..
102 Receive MQTT/JSON messsage requesting to update some parameters (a subset of parameters)
103 call the user callback for each parameters to be updated.
104 return OK (or refused)
105 Publish MQTT/JSON message only with the subset of parameters (updated or not by the user appplication)
106 return OK (from LiveObjectsClient_Cycle)
107 return
Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform

4.7. 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).

4.7.1. Attach a set of collected data

At any moment, application can declare/attach one or many set of 'collected data' to the iotsoftbox-mqtt library by calling the function:

```
int LiveObjectsClient_AttachData(
        uint8_t prefix,
        const char* stream_id,
        const char* model, const char* tags,
        const LiveObjectsD_GpsFix_t* gps_ptr,
        const LiveObjectsD_Data_t* data_ptr, int32_t data_nb);
```

When there is no error, the function returns a handle (positive or null value) of the collected data stream.

In the sample application:

```
appv_hdl_data = LiveObjectsClient_AttachData(
        STREAM_PREFIX, "LO_sample_measures",
        "mV1", "\"Test\"", NULL, appv_set_measures, SET_MEASURES_NB);
```

Where:

```
LiveObjectsD_Data_t appv_set_measures[] = {
        { LOD_TYPE_UINT32, "counter", &appv_measures_counter, 1  },
        { LOD_TYPE_INT32, "temperature", &appv_measures_temp, 1  },
        { LOD_TYPE_FLOAT, "battery_level", &appv_measures_volt, 1  }
};
#define SET_MEASURES_NB (sizeof(appv_set_measures) / sizeof(LiveObjectsD_Data_t))
```

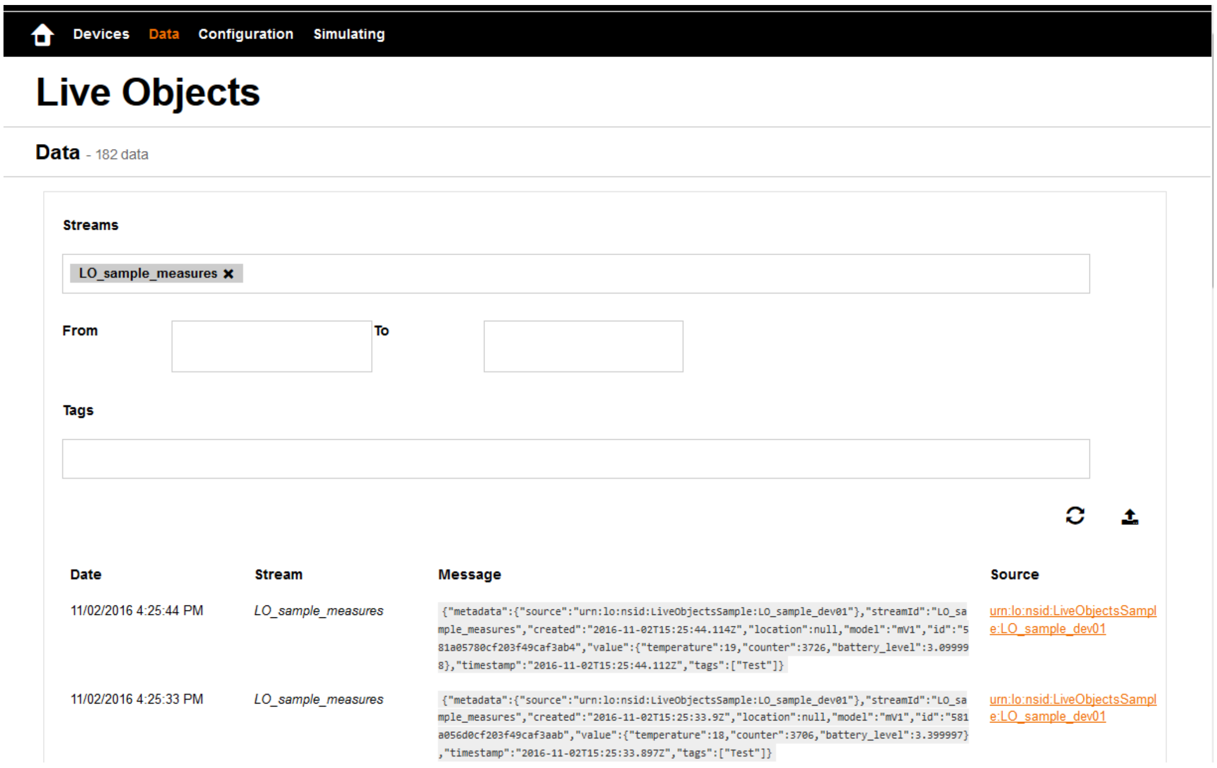
4.7.2. Push the set of collected data

When ‘collected data’ must be *published*, the application must call the `LiveObjectsClient_PushData ( )` function to notify the Datavenue LiveObjects platform (publishing a MQTT/JSON message on the dev/data topic):

```
LiveObjectsClient_PushData(appv_hdl_data);
```

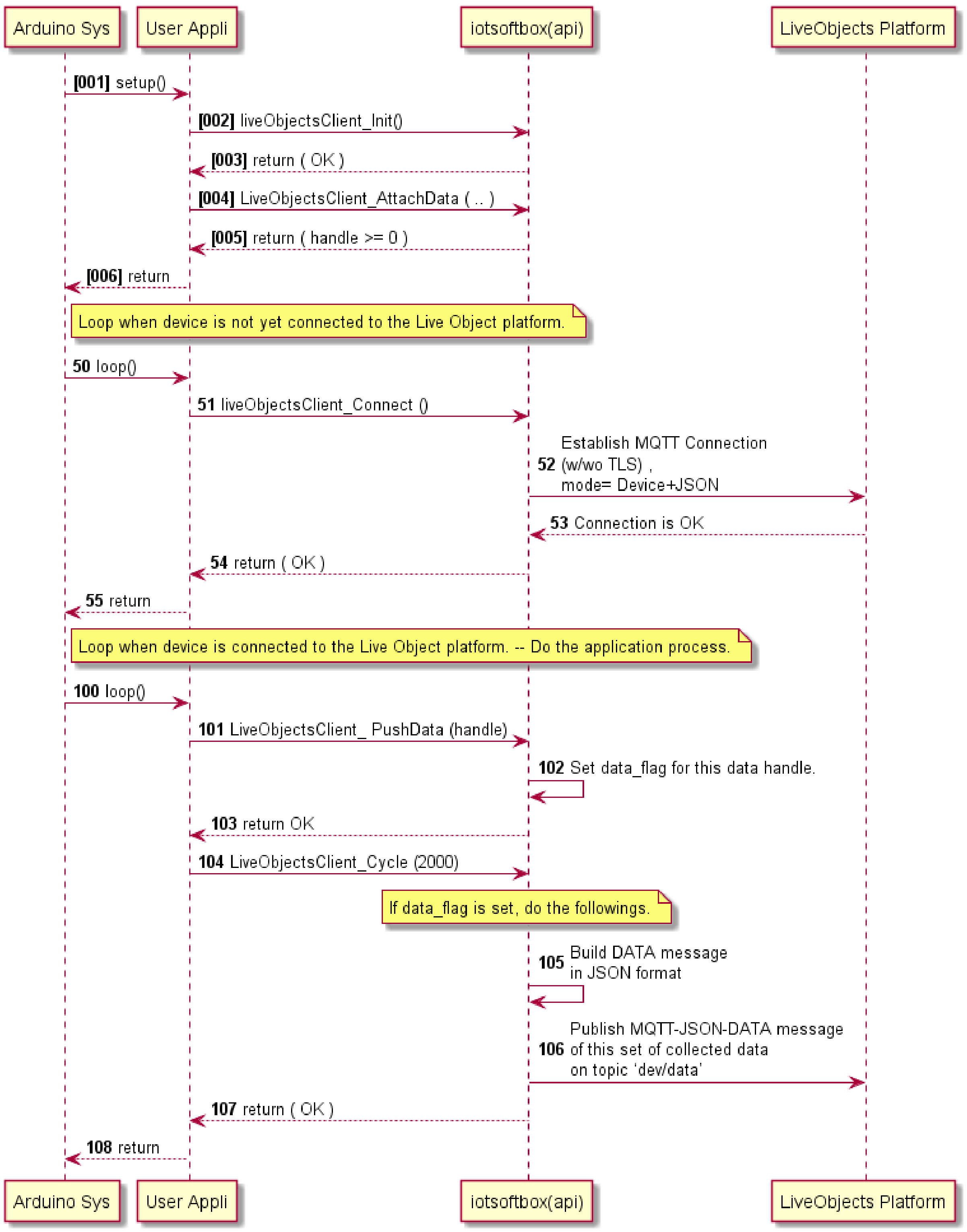
4.7.3. Use of Live Objects Portal to view data stream

On the Datavenue Live Objects portal, the user can check the ‘Collected Data’ published by its connected device (here, filter is set to get only stream = LO_sample_measures):



Date	Stream	Message	Source
11/02/2016 4:25:44 PM	*LO_sample_measures*	`{"metadata":{"source":"urn:lo:nsid:LiveObjectsSample:LO_sample_dev01"},"streamId":"LO_sample_measures","created":"2016-11-02T15:25:44.114Z","location":null,"model":"mV1","id":"581a05780cf203f49caf3ab4","value":{"temperature":19,"counter":3726,"battery_level":3.099998},"timestamp":"2016-11-02T15:25:44.112Z","tags":["Test"]}`	urn:lo:nsid:LiveObjectsSample:LO_sample_dev01
11/02/2016 4:25:33 PM	*LO_sample_measures*	`{"metadata":{"source":"urn:lo:nsid:LiveObjectsSample:LO_sample_dev01"},"streamId":"LO_sample_measures","created":"2016-11-02T15:25:33.9Z","location":null,"model":"mV1","id":"581a056d0cf203f49caf3aab","value":{"temperature":18,"counter":3706,"battery_level":3.399997},"timestamp":"2016-11-02T15:25:33.897Z","tags":["Test"]}`	urn:lo:nsid:LiveObjectsSample:LO_sample_dev01

4.7.4. Sequence Diagram


Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform
[001] setup()
[002] liveObjectsClient_Init()
[003] return (OK)
[004] LiveObjectsClient_AttachData (..)
[005] return (handle >= 0)
[006] return
Loop when device is not yet connected to the Live Object platform.
50 loop()
51 liveObjectsClient_Connect ()
52 Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
53 Connection is OK
54 return (OK)
55 return
Loop when device is connected to the Live Object platform. -- Do the application process.
100 loop()
101 LiveObjectsClient_ PushData (handle)
102 Set data_flag for this data handle.
103 return OK
104 LiveObjectsClient_Cycle (2000)
If data_flag is set, do the followings.
105 Build DATA message in JSON format
106 Publish MQTT-JSON-DATA message of this set of collected data on topic 'dev/data'
107 return (OK)
108 return
Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform

4.8. Commands

4.8.1. Attach a set of commands

At any moment, the application can attach/declare only one set (or group) of ‘commands’ that the device is able to process. For that, the application calls the function:

```
int LiveObjectsClient_AttachCommands(
        const LiveObjectsD_Command_t* cmd_ptr,
        int32_t cmd_nb,
        LiveObjectsD_CallbackCommand_t callback);
```

In the sample application:

```
 ret       =       LiveObjectsClient_AttachCommands(appv_set_commands,       SET_COMMANDS_NB,
main_cb_command)
```

Where:

1. The set of ‘commands’ is defined by an array of `LiveObjectsD_Command_t` elements.
   In the sample application:

```
#define CMD_IDX_RESET 1
#define CMD_IDX_LED 2

LiveObjectsD_Command_t appv_set_commands[] = {
        { CMD_IDX_RESET, "RESET", 0 },
        { CMD_IDX_LED, "LED", 0 }
};
#define SET_COMMANDS_NB (sizeof(appv_set_commands) / sizeof(LiveObjectsD_Command_t))
```

2. The application specifies the callback function (i.e. `commandCb`) which will be called when a command is received from the Live Objects platform.

```
int main_cb_command(LiveObjectsD_CommandRequestBlock_t *pCmdReqBlk) {
        int ret;
        const LiveObjectsD_Command_t *cmd_ptr;

        … // Check input param

        cmd_ptr = pCmdReqBlk->hd.cmd_ptr;

        switch (cmd_ptr->cmd_uref) {
        case CMD_IDX_RESET:  // RESET
                ret = main_cmd_doSystemReset(pCmdReqBlk);
                break;
        case CMD_IDX_LED:  // LED
                ret = main_cmd_doLED(pCmdReqBlk);
                break;
        default:
                ret = -4;
        }
        return ret;
}
```

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

4.8.2. Enable/disable ‘command’ feature

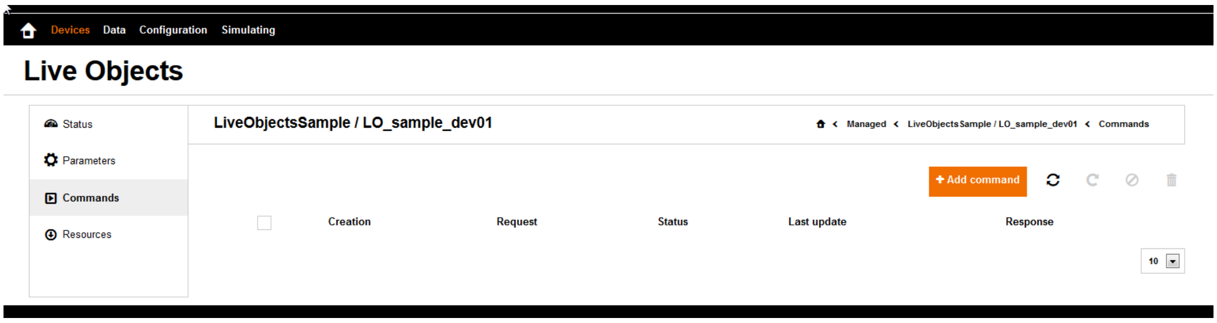
As soon as the device is ready (or not) to process commands, the application can enable (or disable) the ‘command’ feature by calling the function:

```
int LiveObjectsClient_ControlCommands(bool enable);
```

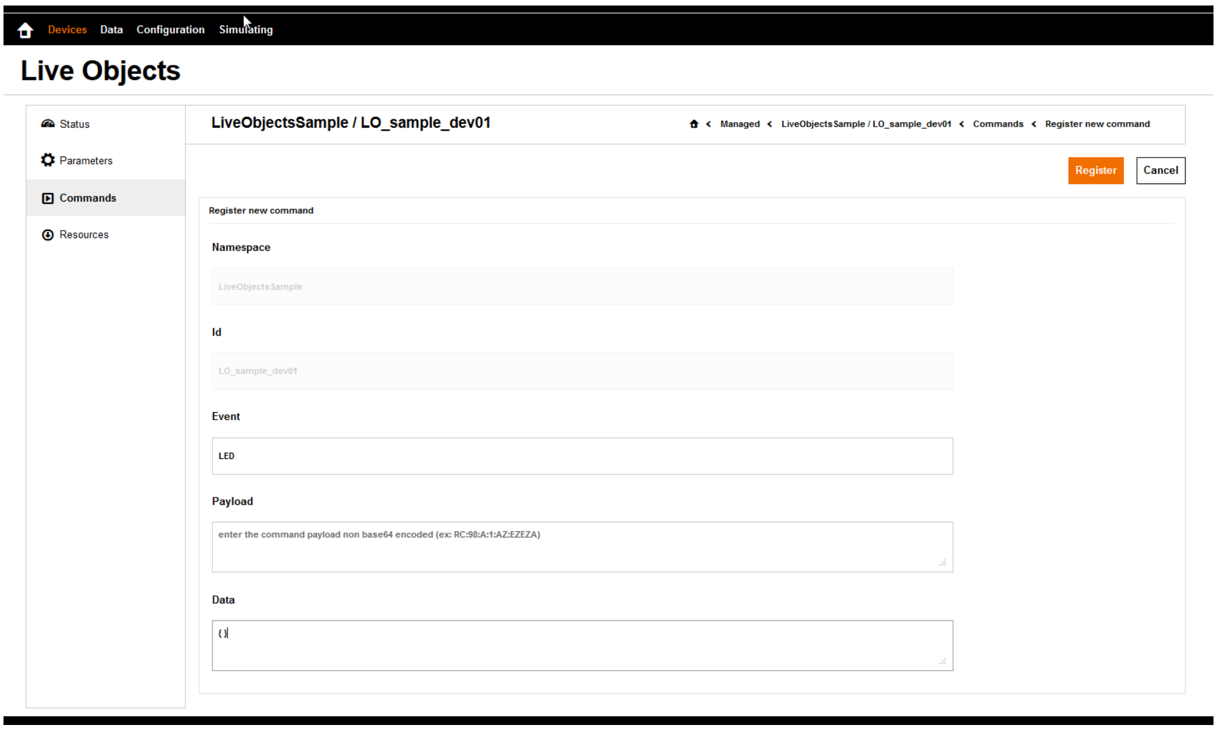
4.8.3. Use of Live Objects Portal to send a command

On the Live Objects Portal,

- Go to tab: Devices -> <your device> -> Commands



- Click on button ‘+ Add command”



Live Objects

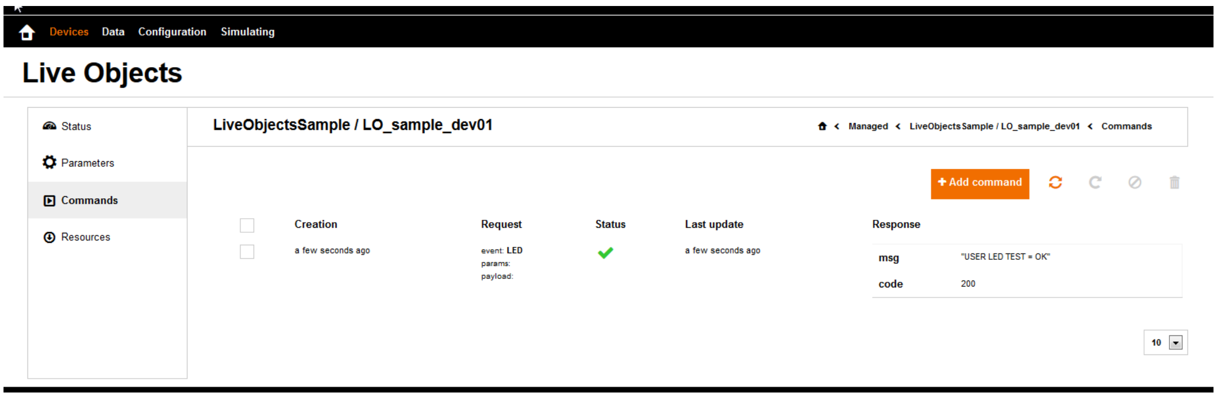
LiveObjectsSample / LO_sample_dev01

Managed < LiveObjectsSample / LO_sample_dev01 < Commands < Register new command

Register new command

Namespace	LiveObjectsSample
Id	LO_sample_dev01
Event	LED
Payload	enter the command payload non base64 encoded (ex: RC:98:A:1:AZ:EZEZA)
Data	{}

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



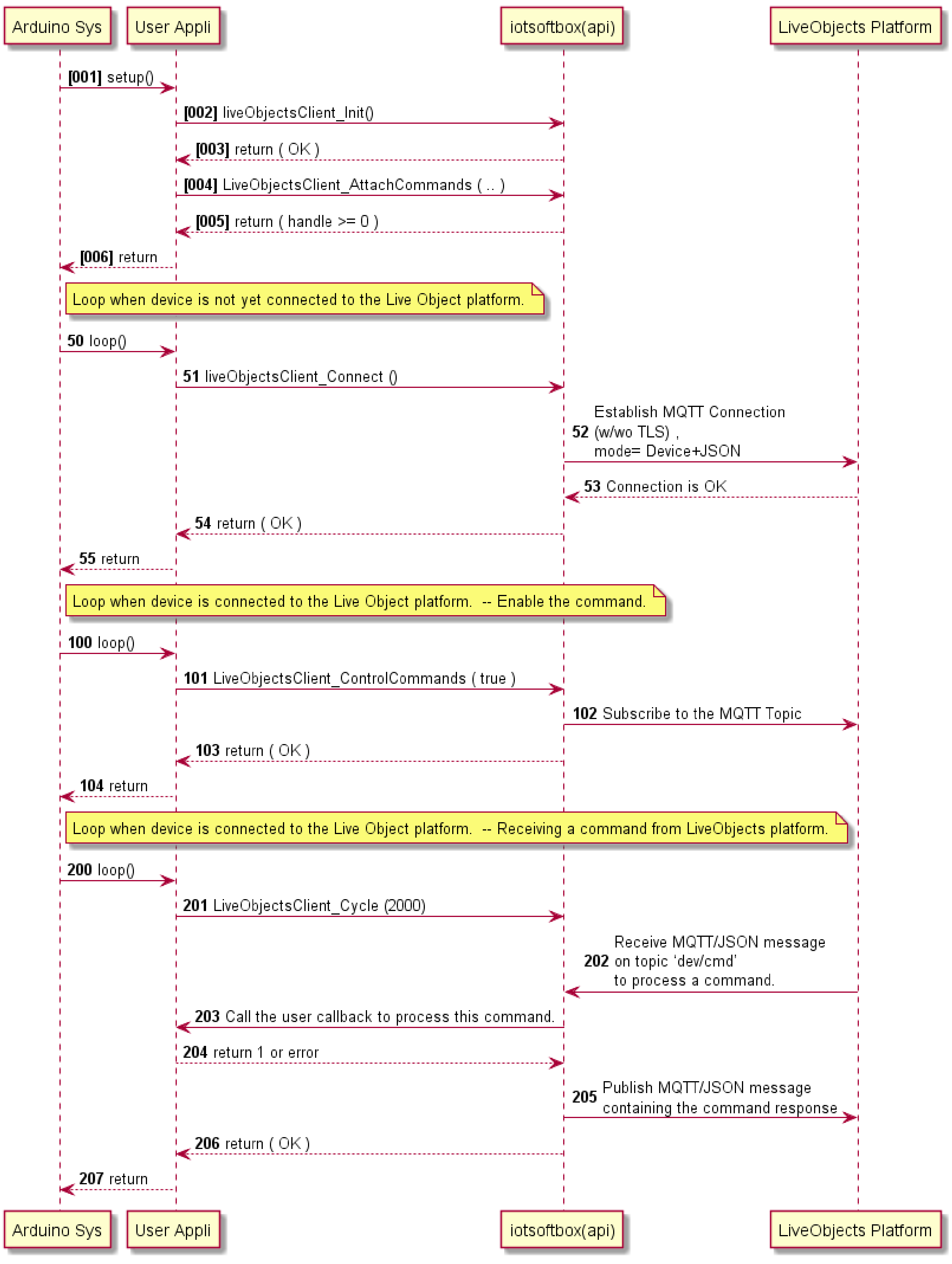
Live Objects

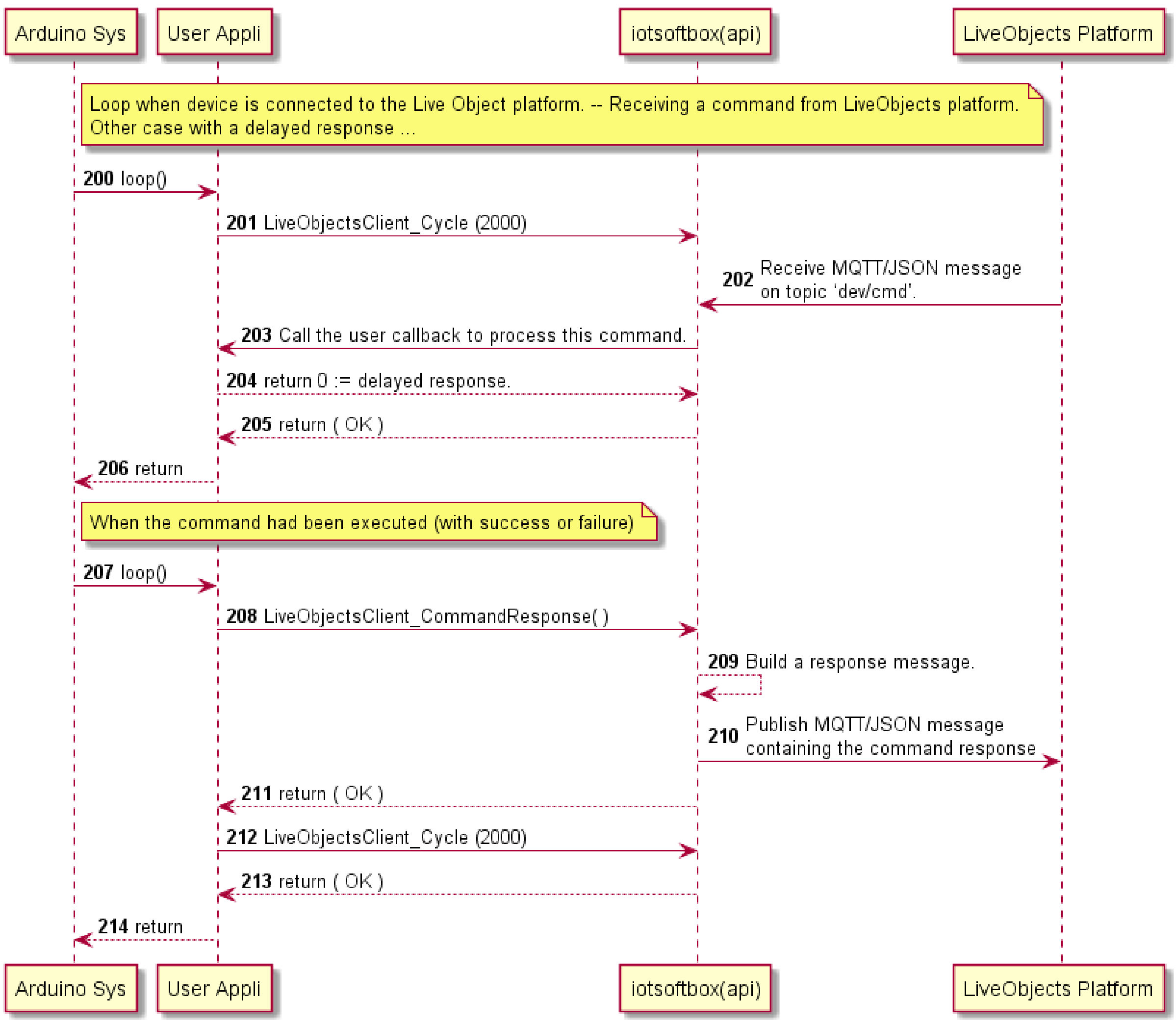
LiveObjectsSample / LO_sample_dev01

Managed < LiveObjectsSample / LO_sample_dev01 < Commands

Creation	Request	Status	Last update	Response
a few seconds ago	event: LED params: payload:	✓	a few seconds ago	msg "USER LED TEST = OK" code 200

4.8.4. Sequence Diagram


Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform
[001] setup()
[002] liveObjectsClient_Init()
[003] return (OK)
[004] LiveObjectsClient_AttachCommands (..)
[005] return (handle >= 0)
[006] return
Loop when device is not yet connected to the Live Object platform.
50 loop()
51 liveObjectsClient_Connect ()
52 Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
53 Connection is OK
54 return (OK)
55 return
Loop when device is connected to the Live Object platform. -- Enable the command.
100 loop()
101 LiveObjectsClient_ControlCommands (true)
102 Subscribe to the MQTT Topic
103 return (OK)
104 return
Loop when device is connected to the Live Object platform. -- Receiving a command from LiveObjects platform.
200 loop()
201 LiveObjectsClient_Cycle (2000)
202 Receive MQTT/JSON message on topic 'dev/cmd' to process a command.
203 Call the user callback to process this command.
204 return 1 or error
205 Publish MQTT/JSON message containing the command response
206 return (OK)
207 return
Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform


Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform
Loop when device is connected to the Live Object platform. -- Receiving a command from LiveObjects platform. Other case with a delayed response ...
200 loop()
201 LiveObjectsClient_Cycle (2000)
202 Receive MQTT/JSON message on topic 'dev/cmd'.
203 Call the user callback to process this command.
204 return 0 := delayed response.
205 return (OK)
206 return
When the command had been executed (with success or failure)
207 loop()
208 LiveObjectsClient_CommandResponse( )
209 Build a response message.
210 Publish MQTT/JSON message containing the command response
211 return (OK)
212 LiveObjectsClient_Cycle (2000)
213 return (OK)
214 return
Arduino Sys
User Appli
iotsoftbox(api)
LiveObjects Platform

4.9. Resources

4.9.1. Attach a set of resources

At any moment, the application can attach/declare only one set (or group) of 'resources' by calling the function:

```
int LiveObjectsClient_AttachResources(
        const LiveObjectsD_Resource_t *rsc_ptr, int32_t rsc_nb,
        LiveObjectsD_CallbackResourceNotify_t ntfyCB,
        LiveObjectsD_CallbackResourceData_t dataCB);
```

In the sample application:

```
ret = LiveObjectsClient_AttachResources(appv_set_resources, SET_RESOURCES_NB,
       main_cb_rsc_ntfy, main_cb_rsc_data);
```

Where:

1. The set of 'resources' is defined by an array of `LiveObjectsD_Resource_t` elements.

   In the sample application:

```
char appv_rv_message[10] = "01.00";
char appv_rv_image[10] = "01.00";

#define RSC_IDX_MESSAGE 1
#define RSC_IDX_IMAGE 2

/// Set of resources
LiveObjectsD_Resource_t appv_set_resources[] = {
       { RSC_IDX_MESSAGE, "message", appv_rv_message, sizeof(appv_rv_message) - 1 },
       { RSC_IDX_IMAGE, "image", appv_rv_image, sizeof(appv_rv_image) - 1 }
};
#define SET_RESOURCES_NB (sizeof(appv_set_resources)/sizeof(LiveObjectsD_Resource_t))
```

2. The application specifies a first callback function (i.e. `commandCb`) called by the iotsoftbox-mqtt library:
   - When a transfer request is received from the Live Objects platform
   - When the transfer is completed (with/without error)

   In the sample application:

```
LiveObjectsD_ResourceRespCode_t main_cb_rsc_ntfy(uint8_t state,
       const LiveObjectsD_Resource_t *rsc_ptr,
       const char *version_old,
       const char *version_new, uint32_t size) {

       LiveObjectsD_ResourceRespCode_t ret = RSC_RSP_OK;  // OK to update the resource

       if ((rsc_ptr) && (rsc_ptr->rsc_uref > 0) && (rsc_ptr->rsc_uref <=
              SET_RESOURCES_NB)) {
              if (state) {
                     if (state == 1) {  // Completed without error
                            …
                     } else { // Completed with error
                            …
                     }
                     appv_rsc_offset = 0;
                     appv_rsc_size = 0;
```

```
                        // Push Status (message has been updated or not)
                        LiveObjectsClient_PushStatus(appv_hdl_status);
                } else {
                        appv_rsc_offset = 0;
                        ret = RSC_RSP_ERR_NOT_AUTHORIZED;
                        switch (rsc_ptr->rsc_uref) {
                        case RSC_IDX_MESSAGE:
                                if (size < (sizeof(appv_status_message) - 1)) {
                                        ret = RSC_RSP_OK;
                                }
                                break;
                        case RSC_IDX_IMAGE:
                                if (size < (sizeof(appv_rsc_image) - 1)) {
                                        ret = RSC_RSP_OK;
                                }
                                break;
                        }
                        if (ret == RSC_RSP_OK) { // Initialize the transfer
                                appv_rsc_size = size;
                        } else { // Transfer is refused
                                appv_rsc_size = 0;
                        }
                }
        } else {
                ret = RSC_RSP_ERR_INVALID_RESOURCE;
        }
        return ret;
}
```

In this callback you can handle the app behavior for each declared resource.

3. The application specifies a second callback function to receive the data from the Live Objects platform.

   In the sample application:

```
int main_cb_rsc_data(const LiveObjectsD_Resource_t *rsc_ptr, uint32_t offset) {
        int ret;

        if (rsc_ptr->rsc_uref == RSC_IDX_MESSAGE) {
                char buf[40];
                if (offset > (sizeof(appv_status_message) - 1)) {
                        return -1;
                }
                ret = LiveObjectsClient_RscGetChunck(rsc_ptr, buf, sizeof(buf) - 1);
                if (ret > 0) {
                        if ((offset + ret) > (sizeof(appv_status_message) - 1)) {
                                return -1;
                        }
                }
        } else if (..) {
                …
        } else {
                ret = -1;
        }
        return ret;
}
```

In this callback, you can receive the data. You can get the data in one time if you have set a big enough buffer or in severals time if you're using a very low capacity device. Once you have the data you can process them.

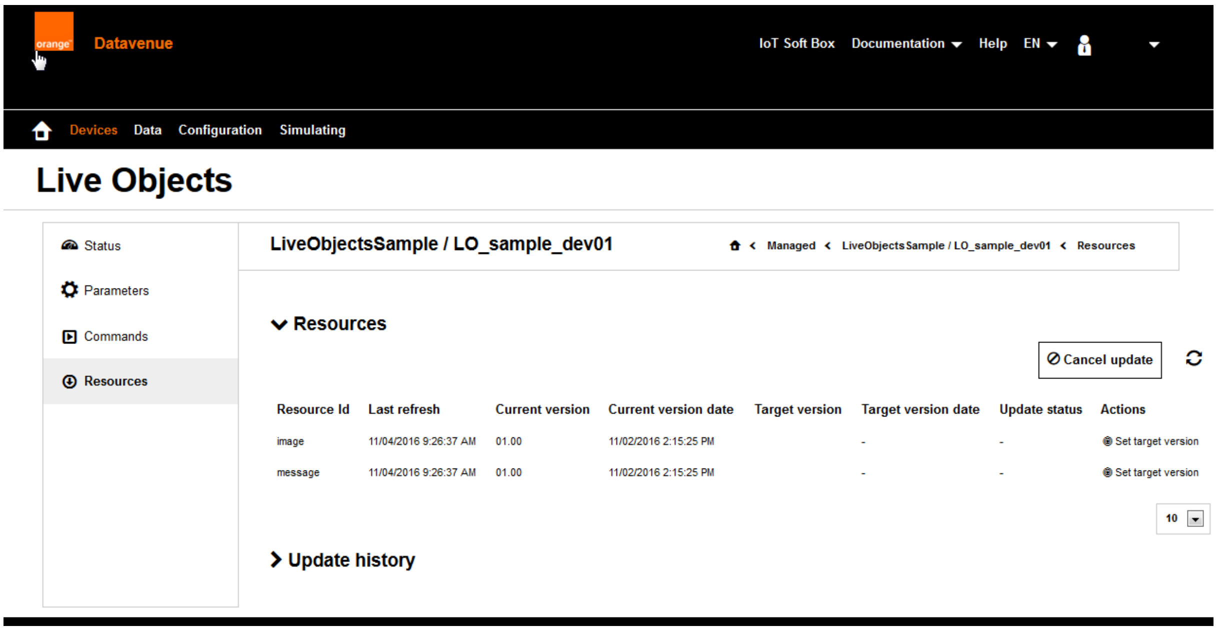
4.9.2. Enable/disable 'resources' feature

As soon as the device is ready (or not) to process the resource update request, the application can enable (or disable) the 'resources' feature by calling the function:

```
int LiveObjectsClient_ControlResources(bool enable);
```

4.9.3. Use of Live Objects Portal to create and update a resource

The first step is to check the list of resources declared by the Live Objects device.



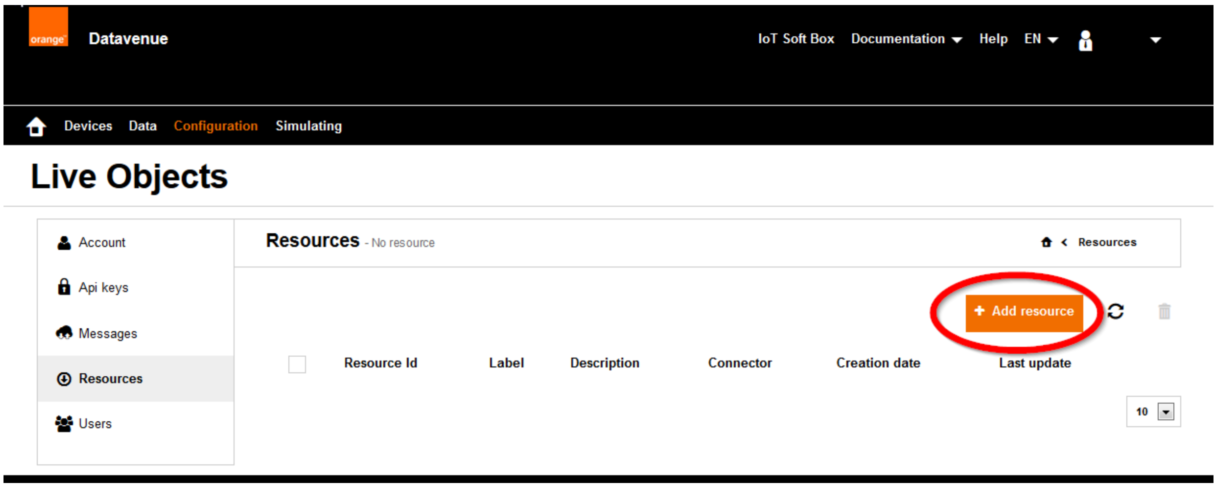
LiveObjectsSample / LO_sample_dev01

Resources

Resource Id	Last refresh	Current version	Current version date	Target version	Target version date	Update status	Actions
image	11/04/2016 9:26:37 AM	01.00	11/02/2016 2:15:25 PM		-	-	Set target version
message	11/04/2016 9:26:37 AM	01.00	11/02/2016 2:15:25 PM		-	-	Set target version

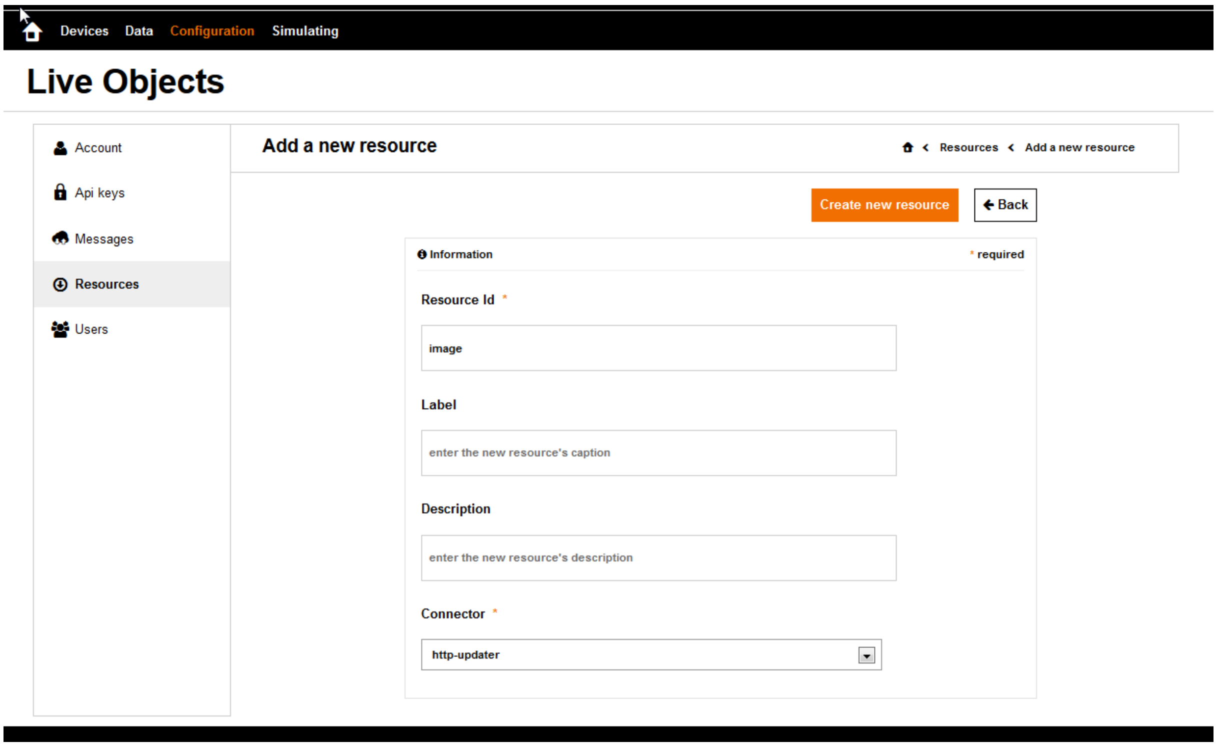
Here, the device 'LiveObjectsSample/LO_sample_dev01' has two resources identified by: *image* and *message*.

Now, the user can create a new resource on the Live Objects platform, in the tab 'Configuration->Resources', associated to these resources



Two fields are mandatory

- Resource Id: set to 'image', resource identified by the device.
- Connector: set to http-updater.



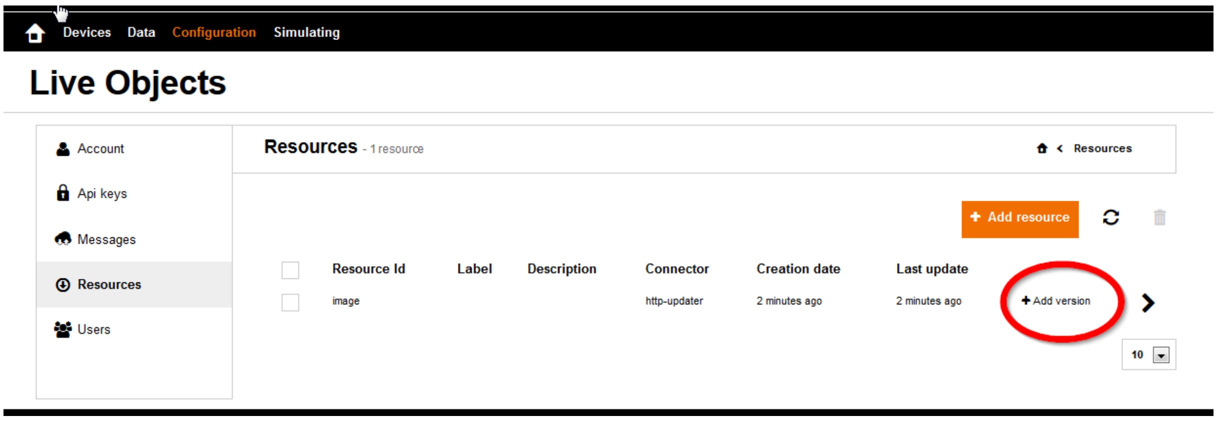
Resource Id *: image

Label: enter the new resource's caption

Description: enter the new resource's description

Connector *: http-updater

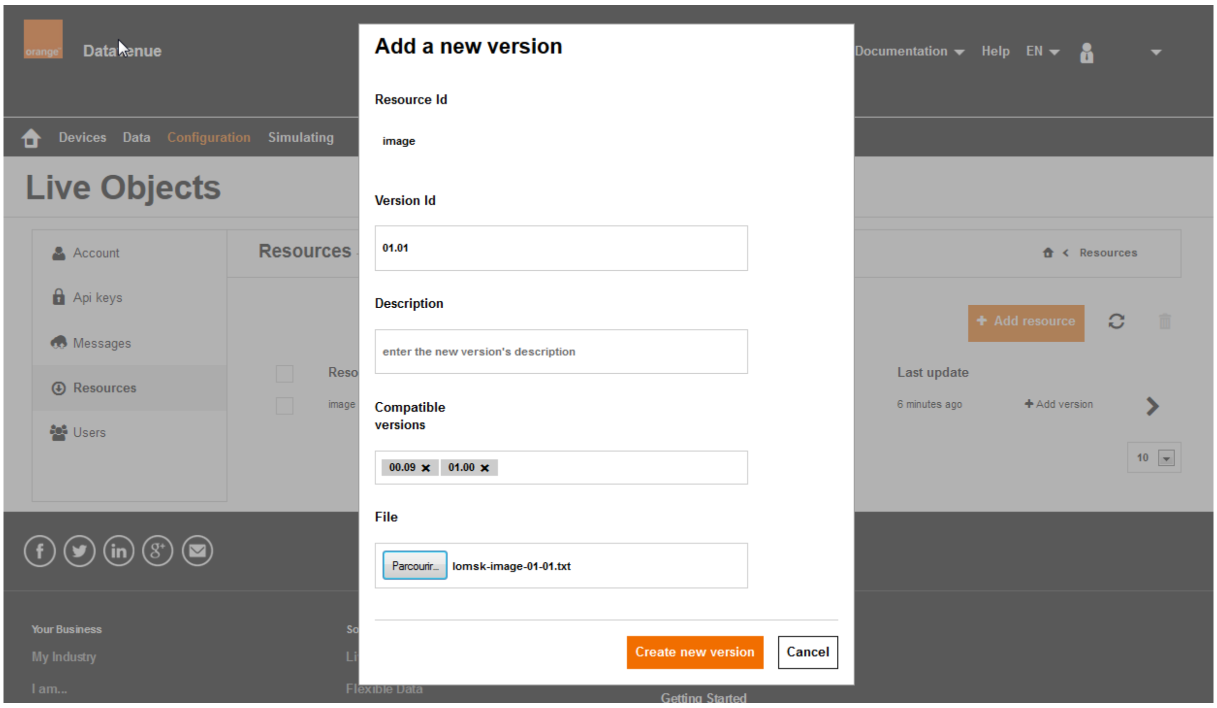
The result is the following:



Resource Id	Label	Description	Connector	Creation date	Last update
image			http-updater	2 minutes ago	2 minutes ago

Then, a new resource version can be attached to this resource 'image', by specifying:

- Version id (i.e. 01.01) for this resource to download on devices
- Compatible versions (optional): the list of current versions deployed on devices which must be able to accept this new version (01.01).
- File, to download on the device



Add a new version

Resource Id
image

Version Id
01.01

Description
enter the new version's description

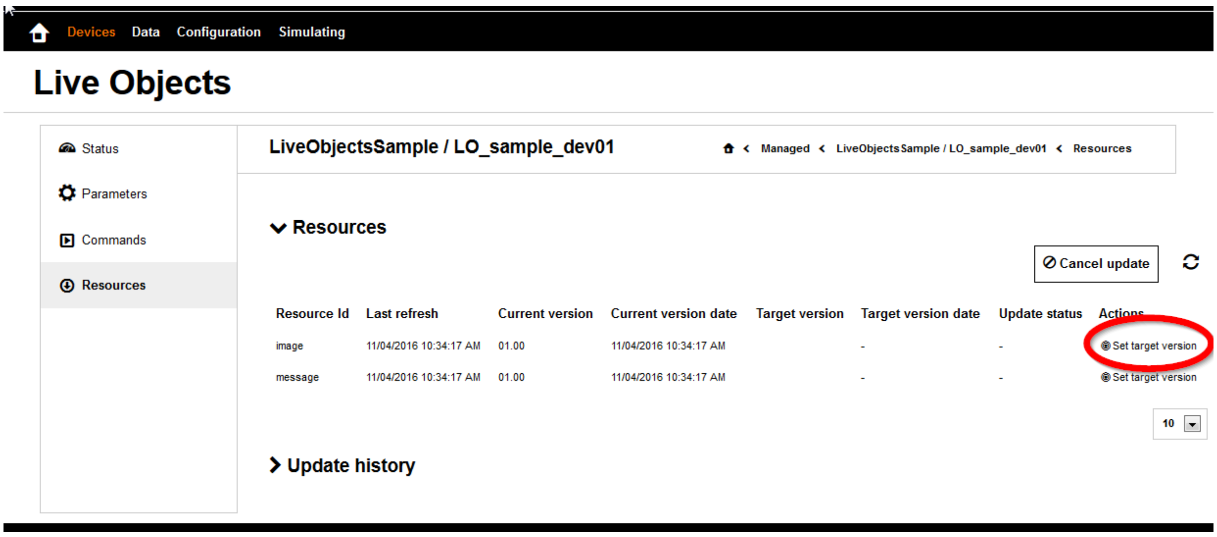
Compatible versions
00.09 ✕ 01.00 ✕

File
Parcourir... lomsk-image-01-01.txt

Create new version | Cancel

Now, the resource update request can be launched for this device:

- Go to the 'devices' tab.
- Select your connected device, here it is "LiveObjectsSample / LO_sample_dev01"
- Go to the 'resources' tab

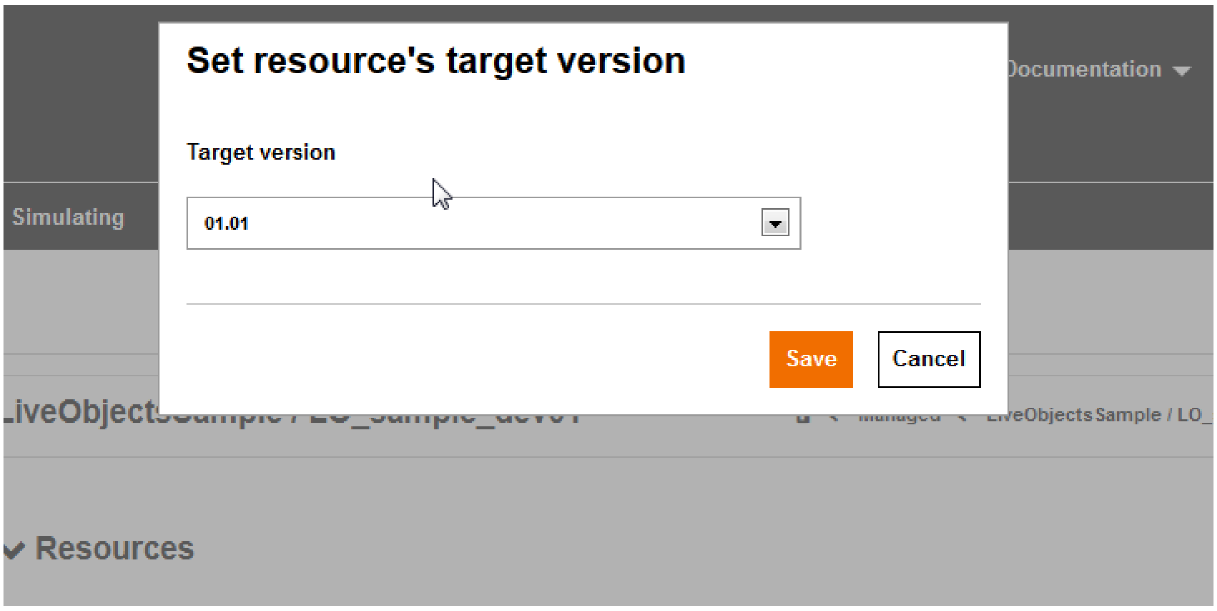


LiveObjectsSample / LO_sample_dev01

Resources

Resource Id	Last refresh	Current version	Current version date	Target version	Target version date	Update status	Actions
image	11/04/2016 10:34:17 AM	01.00	11/04/2016 10:34:17 AM		-	-	Set target version
message	11/04/2016 10:34:17 AM	01.00	11/04/2016 10:34:17 AM		-	-	Set target version

- Click on 'Set target version'
- And select the resource version to download on device

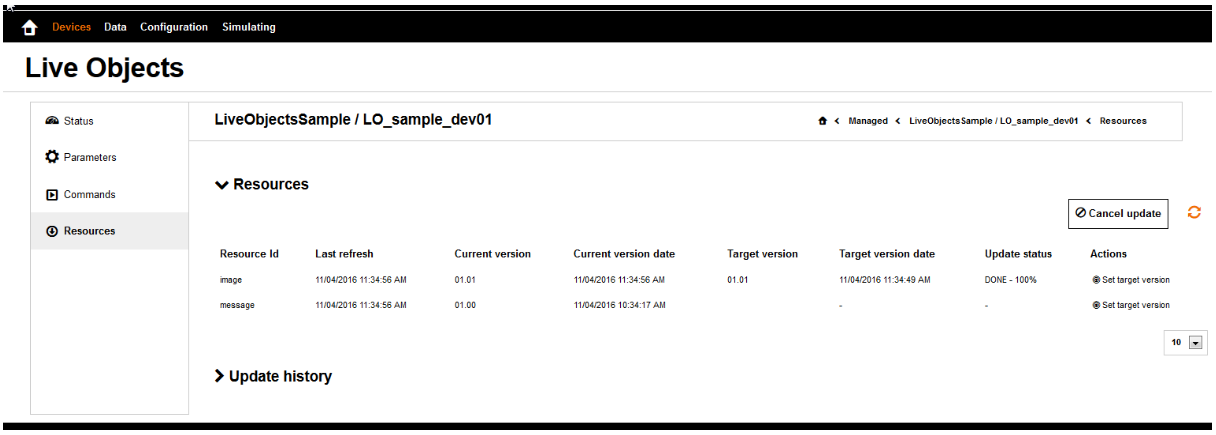


Set resource's target version

Target version

01.01

At the end of transfer, after refreshing the web page, the current version should be equal to the target version:

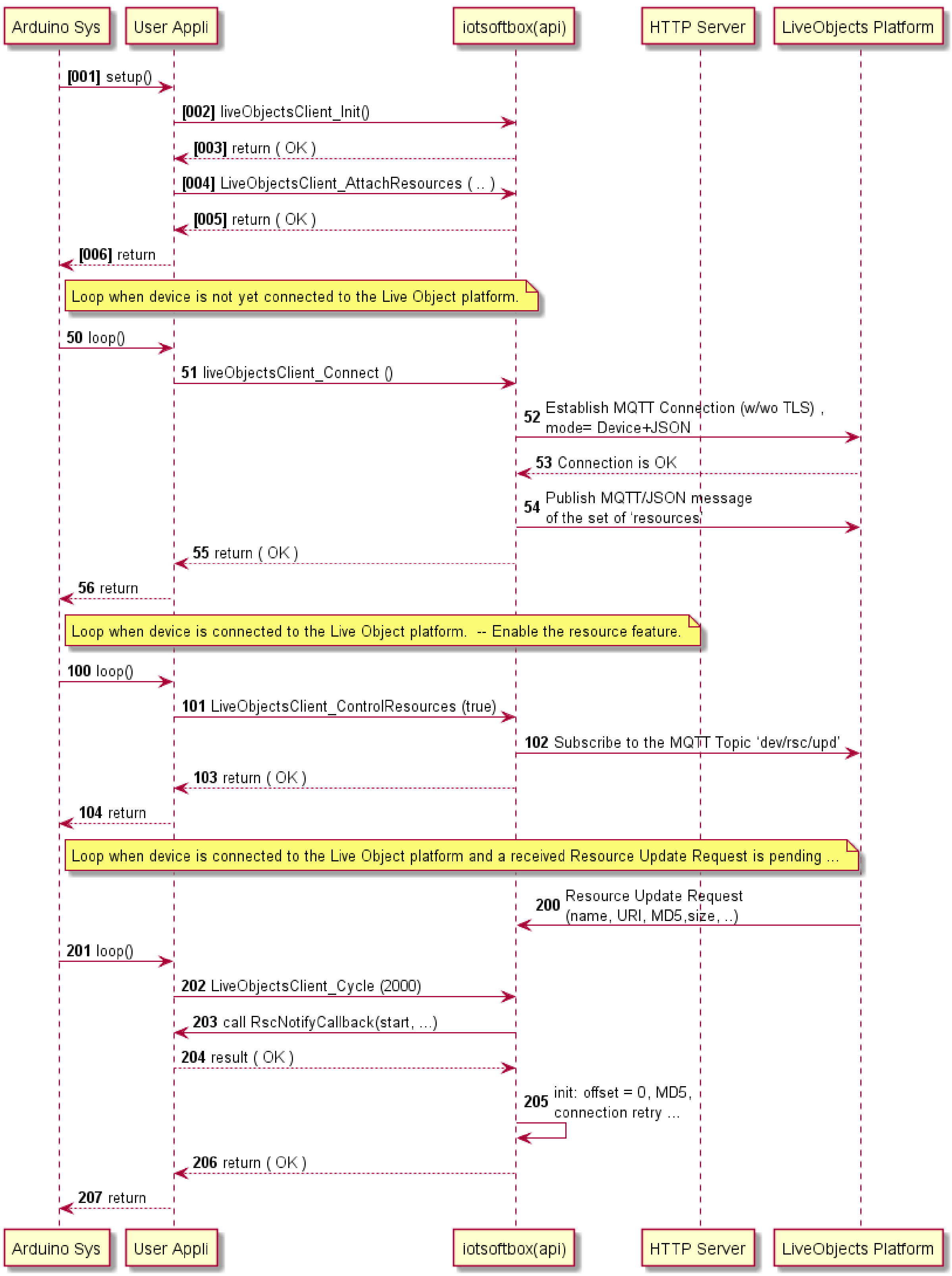


LiveObjectsSample / LO_sample_dev01

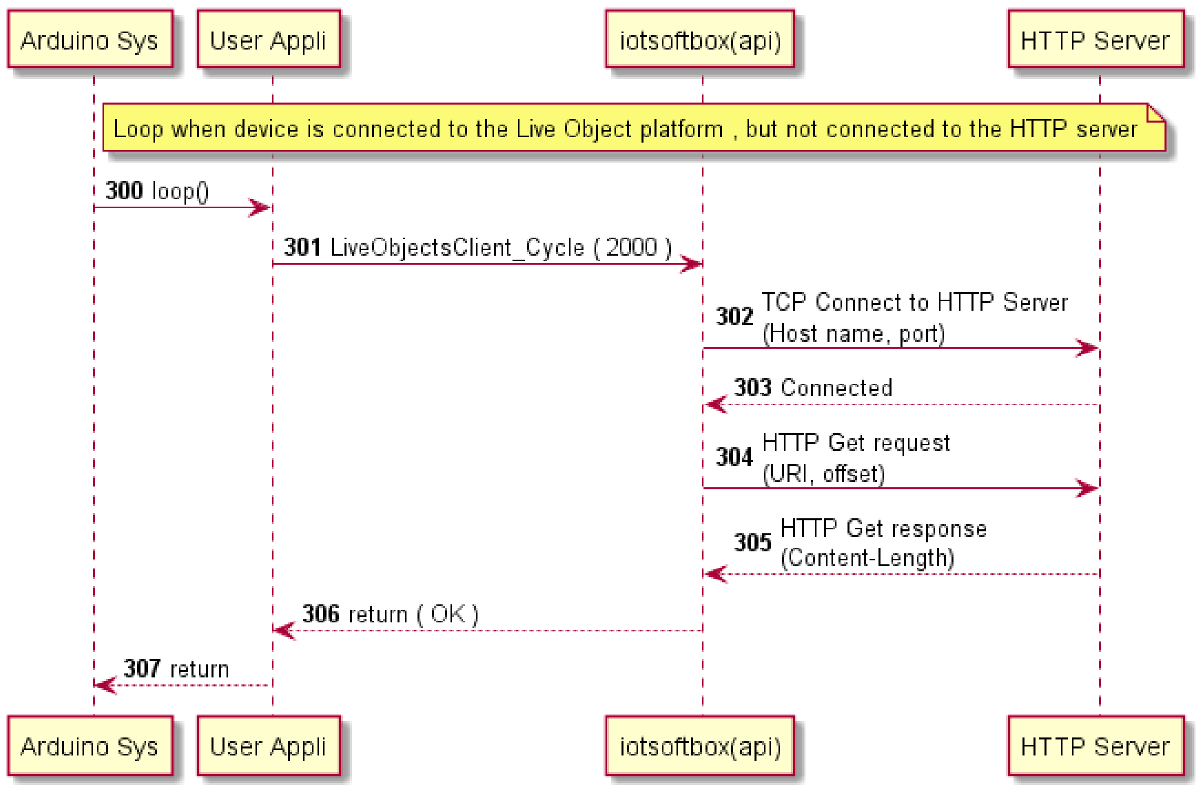
Resources

Resource Id	Last refresh	Current version	Current version date	Target version	Target version date	Update status	Actions
image	11/04/2016 11:34:56 AM	01.01	11/04/2016 11:34:56 AM	01.01	11/04/2016 11:34:49 AM	DONE - 100%	Set target version
message	11/04/2016 11:34:56 AM	01.00	11/04/2016 10:34:17 AM		-	-	Set target version

4.9.4. Sequence diagram


Arduino Sys
User Appli
iotsoftbox(api)
HTTP Server
LiveObjects Platform
[001] setup()
[002] liveObjectsClient_Init()
[003] return (OK)
[004] LiveObjectsClient_AttachResources (..)
[005] return (OK)
[006] return
Loop when device is not yet connected to the Live Object platform.
50 loop()
51 liveObjectsClient_Connect ()
52 Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
53 Connection is OK
54 Publish MQTT/JSON message of the set of 'resources'
55 return (OK)
56 return
Loop when device is connected to the Live Object platform. -- Enable the resource feature.
100 loop()
101 LiveObjectsClient_ControlResources (true)
102 Subscribe to the MQTT Topic 'dev/rsc/upd'
103 return (OK)
104 return
Loop when device is connected to the Live Object platform and a received Resource Update Request is pending ...
200 Resource Update Request (name, URI, MD5,size, ..)
201 loop()
202 LiveObjectsClient_Cycle (2000)
203 call RscNotifyCallback(start, ...)
204 result (OK)
205 init: offset = 0, MD5, connection retry ...
206 return (OK)
207 return

When a resource download is running , first step is to establish a connection to the HTTP server.


Arduino Sys
User Appli
iotsoftbox(api)
HTTP Server
Loop when device is connected to the Live Object platform , but not connected to the HTTP server
300 loop()
301 LiveObjectsClient_Cycle (2000)
302 TCP Connect to HTTP Server (Host name, port)
303 Connected
304 HTTP Get request (URI, offset)
305 HTTP Get response (Content-Length)
306 return (OK)
307 return
Arduino Sys
User Appli
iotsoftbox(api)
HTTP Server

An example of URI is:

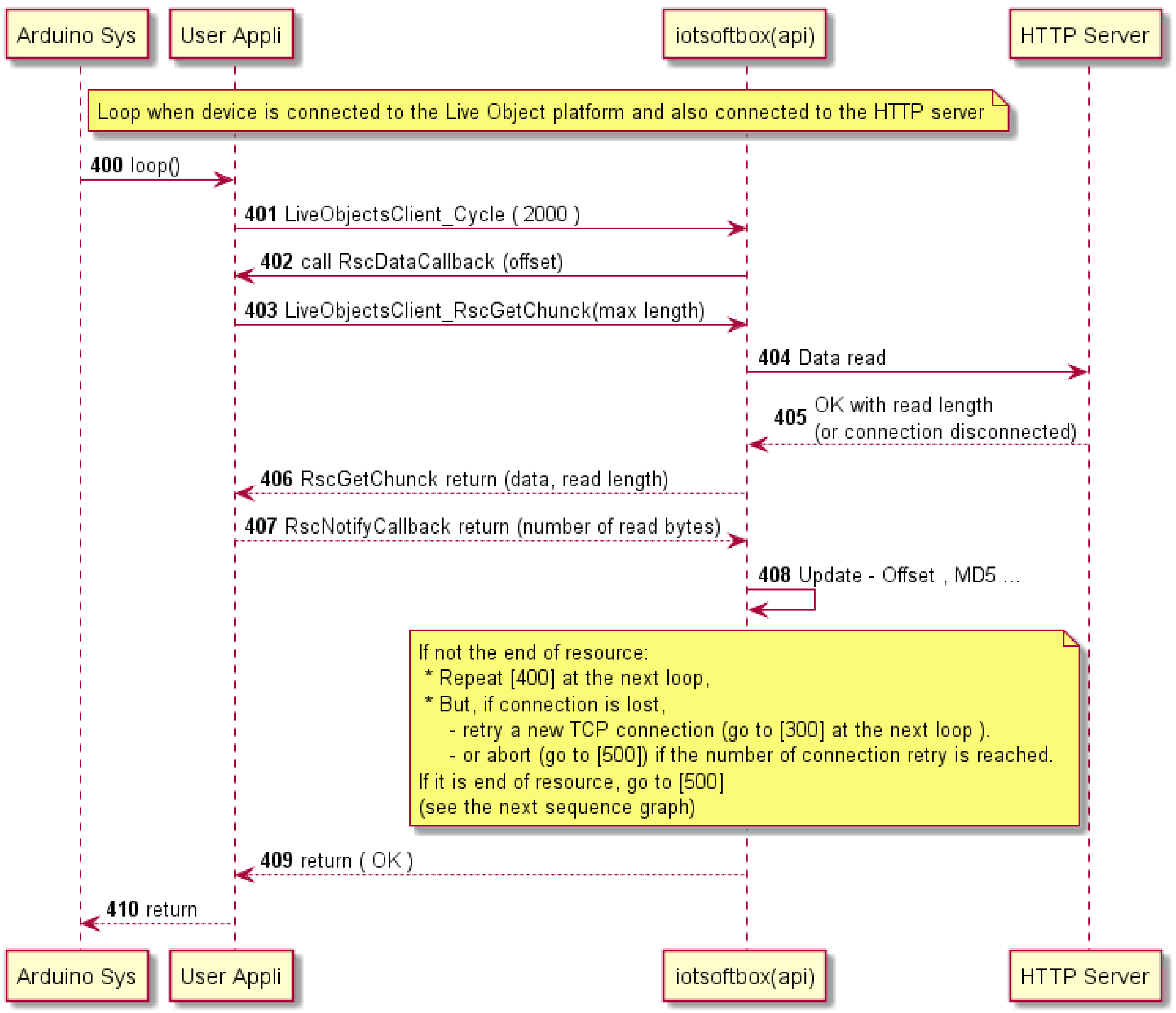
    http://liveobjects.orange-business.com:80/dl/18p1bj775jhk0pj6p49076hk45

And the header of HTTP Get Response is similar to:

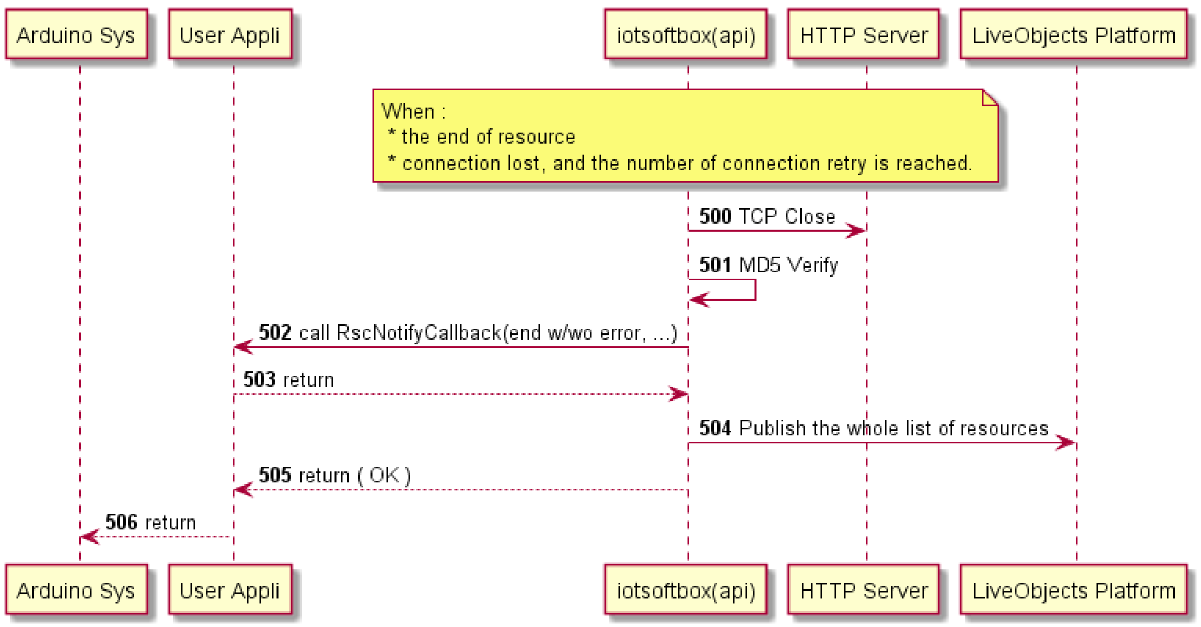
```
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Fri, 04 Nov 2016 10:34:50 GMT
Content-Type: application/force-download; charset=UTF-8
Content-Length: 1974
Connection: close
X-Application-Context: lo-http-updater:prod:8080
Access-Control-Allow-Headers: X-Requested-With, Content-Type
Access-Control-Allow-Credentials: true
```

Then, at each next loop, while the resource transfer is running, get data from the HTTP server …

Note that the user application provides the data buffer to retrieve data from the TCP connection.


Arduino Sys
User Appli
iotsoftbox(api)
HTTP Server
Loop when device is connected to the Live Object platform and also connected to the HTTP server
400 loop()
401 LiveObjectsClient_Cycle (2000)
402 call RscDataCallback (offset)
403 LiveObjectsClient_RscGetChunck(max length)
404 Data read
405 OK with read length (or connection disconnected)
406 RscGetChunck return (data, read length)
407 RscNotifyCallback return (number of read bytes)
408 Update - Offset , MD5 ...
If not the end of resource:
* Repeat [400] at the next loop,
* But, if connection is lost,
- retry a new TCP connection (go to [300] at the next loop).
- or abort (go to [500]) if the number of connection retry is reached.
If it is end of resource, go to [500]
(see the next sequence graph)
409 return (OK)
410 return
Arduino Sys
User Appli
iotsoftbox(api)
HTTP Server

And when the resource transfer is completed (with success or failure), do the following:


Arduino Sys
User Appli
iotsoftbox(api)
HTTP Server
LiveObjects Platform
When :
* the end of resource
* connection lost, and the number of connection retry is reached.
500 TCP Close
501 MD5 Verify
502 call RscNotifyCallback(end w/wo error, ...)
503 return
504 Publish the whole list of resources
505 return (OK)
506 return

5. Additional Information

5.1. Doxygen documentation

The iotsoftbox-mqtt library is documented using the Doxygen source code comments (mainly for the '*public*' header files).

To generate the documentation, download doxygen , then :

1. Launch the following command in a command window:

```
cd <user arduino directory>libraries\iotsoftbox_mqtt_arduino\doxygen
doxygen.exe  liveobjects_iotsoftbox.doxyfile
```

2. Or you can also use a GUI : doxywizard

   Step 1: File->Open .. : select the file
   iotsoftbox_mqtt_arduino\doxygen\liveobjects_iotsoftbox.doxyfile

   Step 2: Go to the **Run** tab and press **Run Doxygen.**

The doc will be generate into an apidoc folder. Open `apidoc/html/index.html` to start browsing the doc.

5.2. Debug

The iotsoftbox-mqtt library uses MACRO definitions to print traces. Theses MACROs are defined in the **loc_trace.h** header file depending on platform.

- `LOTRACE_ERR`
- `LOTRACE_WARN`
- `LOTRACE_NOTICE`
- `LOTRACE_INF`
- `LOTRACE_DBG1`
- `LOTRACE_DBG2`
- `LOTRACE_DBG_VERBOSE`

5.3. IoT Soft Box Library Configuration

The iotsoftbox-mqtt library can be tuned according to the target and/or application constraints (memory, network, use or not of Live Objects features...) All tunable parameters are defined with theirs default values in the header file **liveobjects-client/LiveObjectsClient_Config.h**.

Then, the user application can overwrite theses values in the header file **liveobjects_dev_config.h**

Tunable parameters are:

- `LOC_MQTT_DUMP_MSG`
  Dump MQTT message - set to 1 = text only, 2 = hexa only, 3 = text+hexa
- `LOC_FEATURE_LO_STATUS`
  Support or not the Live Objects 'Status' feature.
- `LOC_FEATURE_LO_PARAMS`
  Support or not the Live Objects 'Configuration Parameters' feature.
- `LOC_FEATURE_LO_DATA`
  Support or not the Live Objects 'Collected Data' feature.
- `LOC_FEATURE_LO_COMMANDS`
  Support or not the Live Objects 'Commands' feature.
- `LOC_FEATURE_LO_RESOURCES`
  Support or not the Live Objects 'resources' feature.

- `LOC_MQUEUE`
  Implement or not the message queue (default: 1)
- `LOM_PUSH_ASYNC`
  Enable or not the '*asynchronous*' mode. (default 0).
  Set to 1 when board has a small size of RAM. Therefore
    - No memory allocation
    - No message queue

- `LOC_SERV_TIMEOUT`
  Connection Timeout in milliseconds (default 20 seconds)
- `LOC_MQTT_API_KEEPALIVEINTERVAL_SEC`
  Period of MQTT Keepalive message (default: 30 seconds)
- `LOC_MQTT_DEF_COMMAND_TIMEOUT`
  Timeout in milliseconds to wait for a MQTT ACK/NACK response after sending MQTT request (default: 5 seconds)

- `LOC_MQTT_DEF_SND_SZ`
  Size (in bytes) of static MQTT buffer used to send a MQTT message (default: 2 K bytes)
- `LOC_MQTT_DEF_RCV_SZ`
  Size (in bytes) of static MQTT buffer used to receive a MQTT message (default: 2 K bytes)

- `LOC_MQTT_DEF_TOPIC_NAME_SZ`
  Max Size (in bytes) of MQTT Topic name (default: 40 bytes)
- `LOC_MQTT_DEF_DEV_ID_SZ`
  Max Size (in bytes) of Device Identifier (default: 20 bytes)
- `LOC_MQTT_DEF_NAME_SPACE_SZ`
  Max Size (in bytes) of Name Space (default: 20 bytes)
- `LOC_MQTT_DEF_PENDING_MSG_MAX`
  Max Number of pending MQTT Publish messages (default: 5 messages)
- `LOC_MAX_OF_COMMAND_ARGS`
  Max Number of arguments in command (default: 5 arguments)

- `LOC_MAX_OF_STATUS_SET`
  Max Number of status sets (default: 1)
- `LOC_MAX_OF_DATA_SET`
  Max Number of collected data streams (or also named 'data sets') (default: 5 data streams)

- `LOM_SETOFDATA_STREAM_ID_SZ`
  Max Size (in bytes) of Data stream Id (default: 80 bytes)
- `LOM_SETOFDATA_MODEL_SZ`

  Max Size (in bytes) of Model defined in a data stream (default: 80 bytes).
  If set to 0, no model field.
- LOM_SETOFDATA_TAGS_SZ
  Max Size (in bytes) of Data Tags defined in a data stream (default: 80 bytes).
  If set to 0, no tags field.

- LOM_JSON_BUF_SZ
  Size (in bytes) of static JSON buffer used to encode the JSON payload to be sent (default: 1 K bytes)
- LOM_JSON_BUF_USER_SZ
  Size (in bytes) of static JSON buffer used to encode a user JSON payload (default: 1 K bytes)
  Note: this buffer is not instantied when the MQUEUE is not implemented.