



IoT Soft Box Starter Kit

User Manual for `iotsoftbox-mqtt` library

Arduino Edition

Table of contents

1. INTRODUCTION.....	4
1.1. Document purpose	4
1.2. Reference documents	4
2. OVERVIEW	5
2.1. What is Live Objects?.....	5
2.2. Arduino	5
2.3. IoT Soft Box	5
3. GETTING STARTED.....	7
3.1. Hardware Environment and compatibility	7
3.2. Access to Live Objects.....	7
3.2.1. Account creation	7
3.2.2. Log in	10
3.2.3. API Key creation	11
3.3. Live Objects IoT examples using iotsoftbox-mqtt library	12
3.3.1. Introduction	12
3.3.2. Packages dependences	13
3.3.3. Configure workstation.....	13
3.3.4. Build	14
3.3.5. Launch	14
4. DETAILED FEATURES	17
4.1. General	17
4.2. Connectivity	17
4.3. Device.....	18
4.4. Thread Models: Multi-thread or single thread.	19
4.5. Status	19
4.5.1. Attach a set of 'status' data.....	19
4.5.2. Push a set of 'status' data	20
4.5.3. Use of Live Objects portal to view/check the set of status	20
4.5.4. Sequence Diagram	21
4.6. Parameters	23
4.6.1. Attach a set of configuration parameters	23
4.6.2. Push a set of configuration parameters.....	25
4.6.3. Use of Live Objects Portal to set/change parameters	25
4.6.4. Sequence Diagram	26
4.7. Collected Data	26
4.7.1. Attach a set of collected data	27
4.7.2. Push the set of collected data	28
4.7.3. Use of Live Objects Portal to view data stream	28
4.7.4. Sequence Diagram	29
4.8. Commands	30



4.8.1.	Attach a set of commands.....	30
4.8.2.	Enable/disable 'command' feature	31
4.8.3.	Use of Live Objects Portal to send a command	31
4.8.4.	Sequence Diagram	32
4.9.	Resources.....	34
4.9.1.	Attach a set of resources	34
4.9.2.	Enable/disable 'resources' feature.....	37
4.9.3.	Use of Live Objects Portal to create and update a resource	38
4.9.4.	Sequence diagram	42
5.	ADDITIONAL INFORMATION	46
5.1.	Doxygen documentation	46
5.2.	Debug	46
5.3.	IoT Soft Box Library Configuration.....	47

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 (1.4.1.)



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.

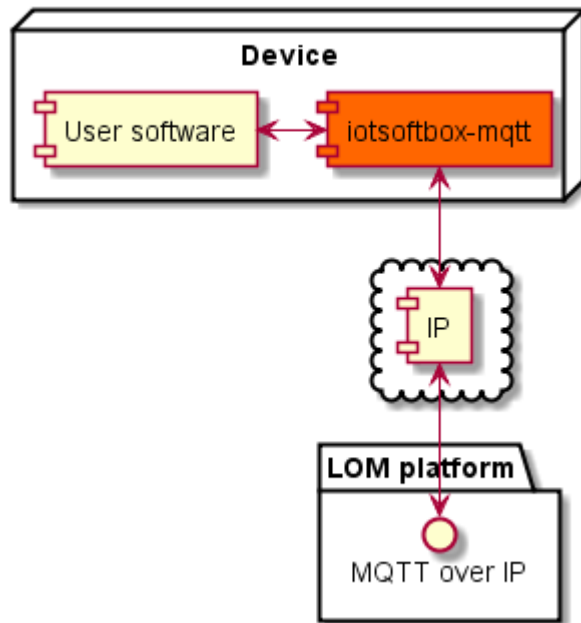


Figure 1 – IoT Soft Box integration in a system

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.

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](#).
- [Mbed TLS](#) is used to include cryptographic and SSL/TLS capabilities in embedded devices.

3. Getting started

3.1. Hardware Environment and compatibility

As for today, we are compatible with the following equipments:

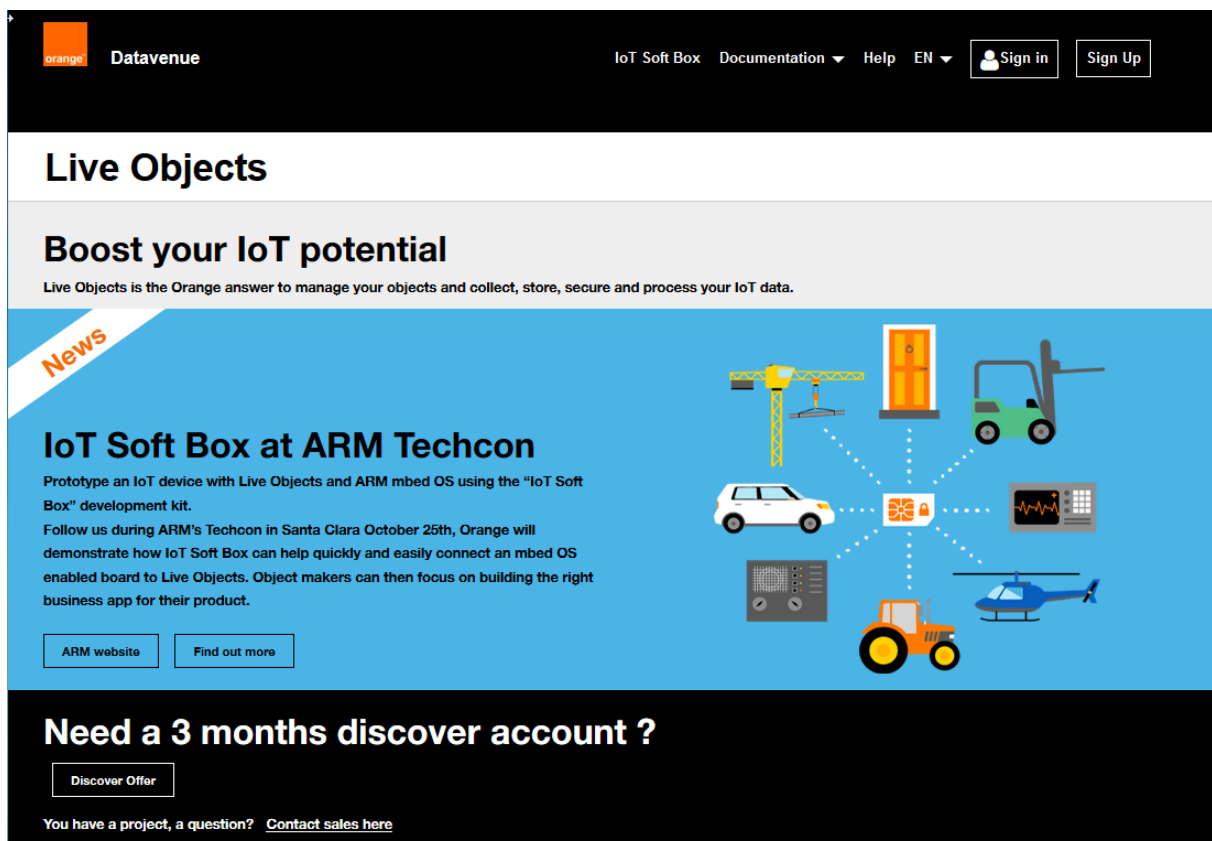
- [Mediatek LinkIt One](#) : Other cards may be compatible but were not tested.

3.2. Access to Live Objects

3.2.1. Account creation

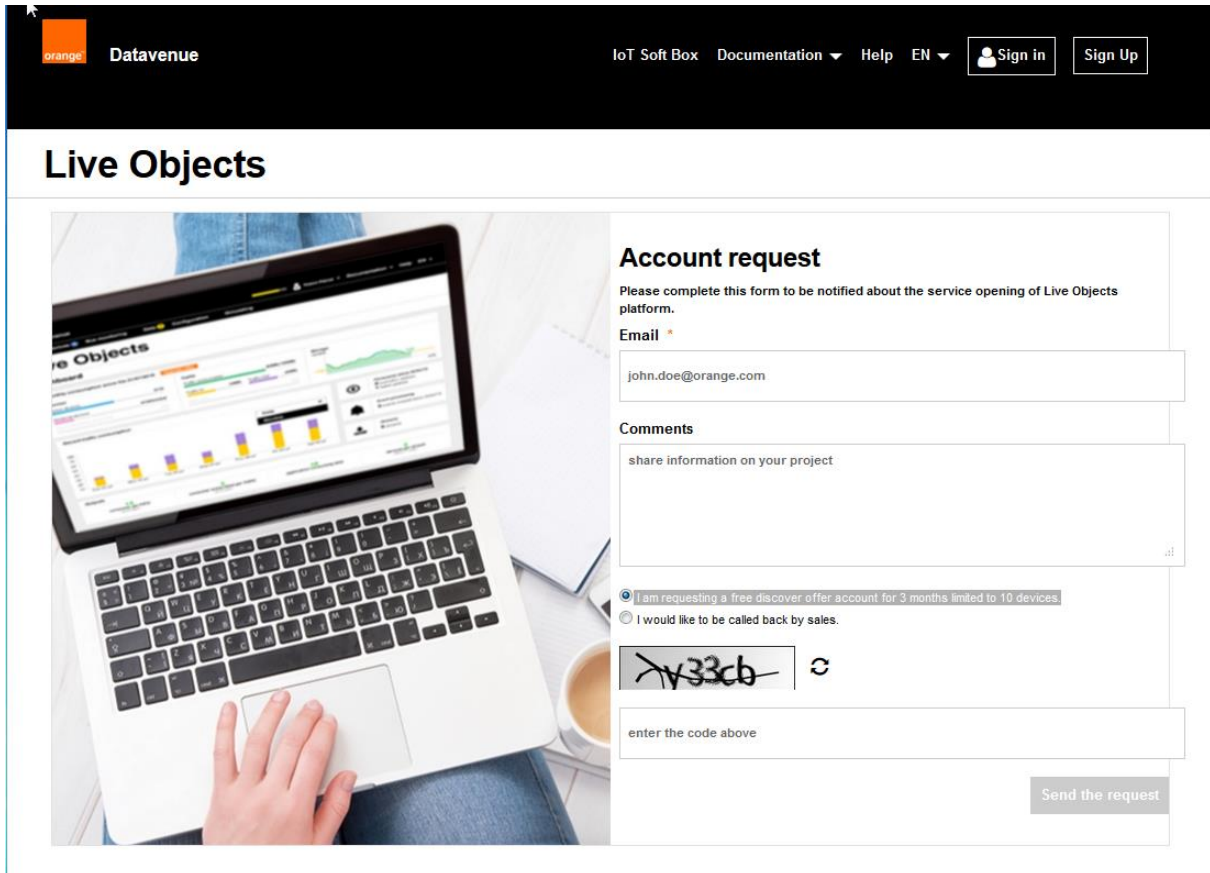
In order to use Live Objects, you need to have a dedicated account on the service.

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



The screenshot shows the Orange Live Objects portal homepage. At the top is a navigation bar with the Orange logo, 'Datavenue', and links for 'IoT Soft Box', 'Documentation', 'Help', 'EN', 'Sign in', and 'Sign Up'. Below the navigation bar is a large banner with the title 'Live Objects' and the subtitle 'Boost your IoT potential'. The banner text states: 'Live Objects is the Orange answer to manage your objects and collect, store, secure and process your IoT data.' A 'News' section on the left highlights 'IoT Soft Box at ARM Techcon', mentioning a prototype device and a demonstration at ARM's Techcon in Santa Clara on October 25th. To the right of the text is a diagram showing various IoT devices (car, crane, forklift, tractor, helicopter, server, and a central IoT icon) connected by dotted lines. At the bottom of the banner is a dark section titled 'Need a 3 months discover account ?' with a 'Discover Offer' button and a link to '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.

7v33cb

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.

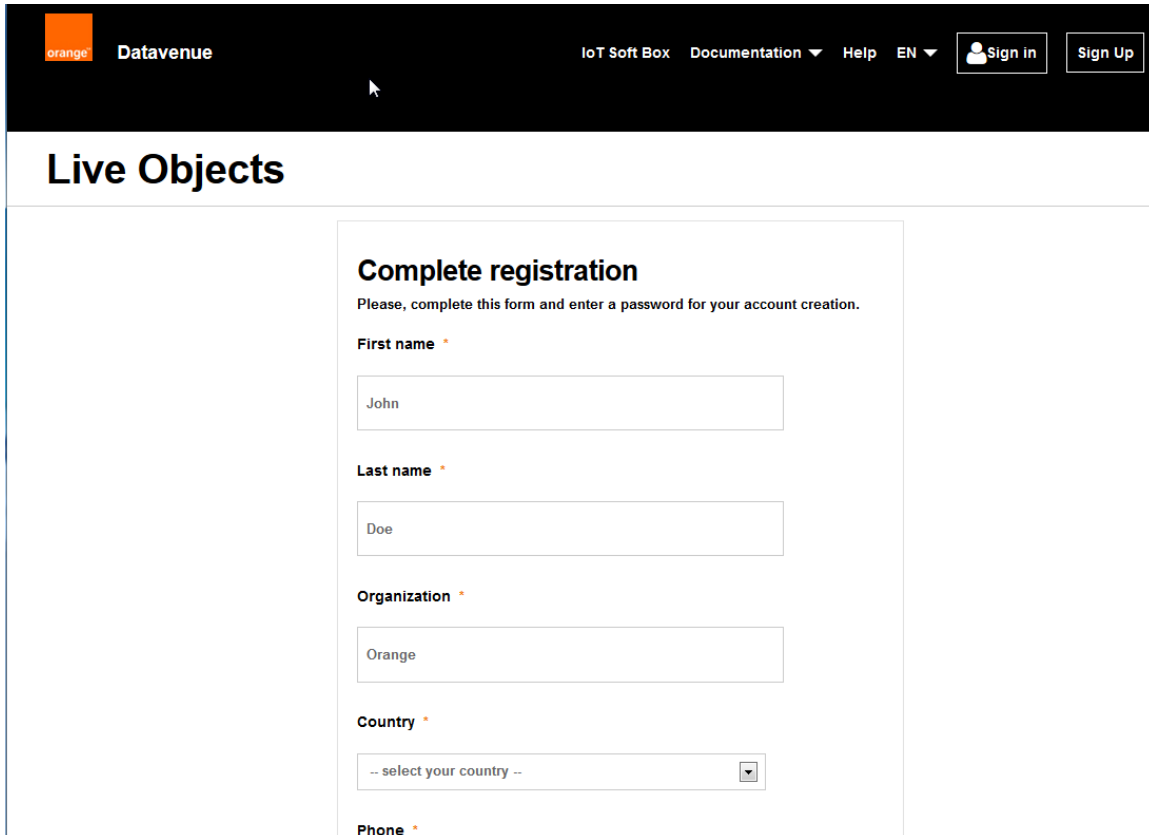
['john' Account activation](#)

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'.

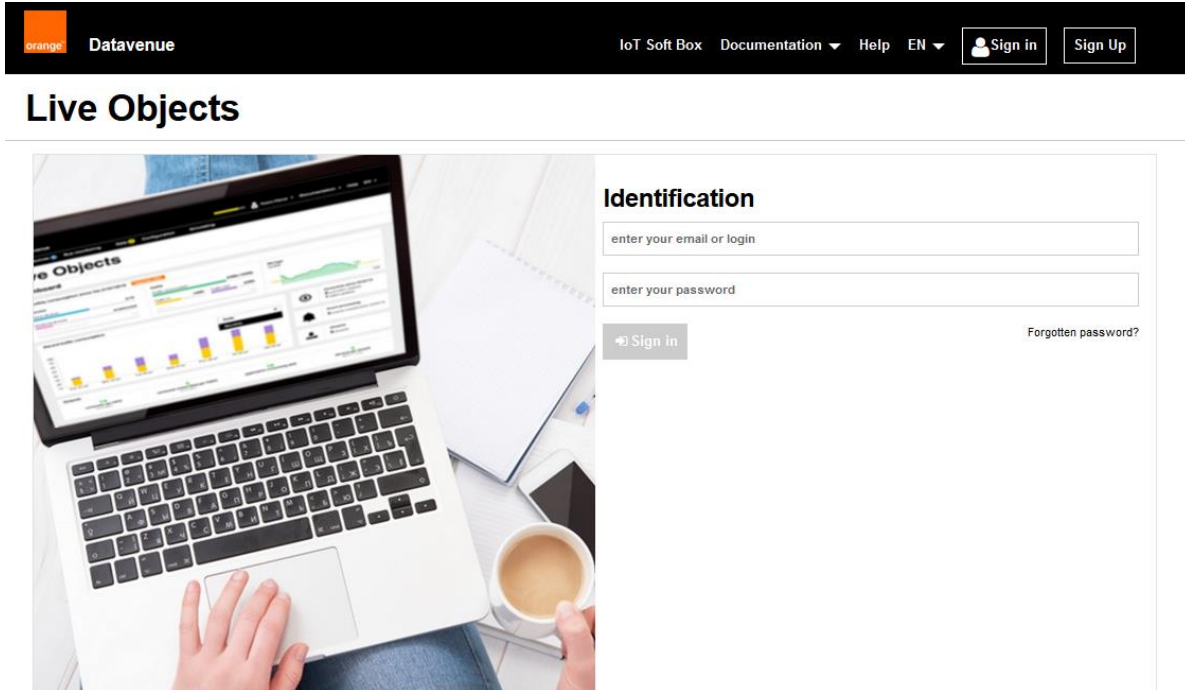


The screenshot shows the Datavenue Live Objects portal. The header is black with the Orange logo and 'Datavenue' on the left, and 'IoT Soft Box', 'Documentation', 'Help', 'EN', 'Sign in', and 'Sign Up' on the right. The main heading is 'Live Objects'. Below it is a registration form titled 'Complete registration' with the instruction 'Please, complete this form and enter a password for your account creation.' The form fields are: 'First name' (filled with 'John'), 'Last name' (filled with 'Doe'), 'Organization' (filled with 'Orange'), 'Country' (a dropdown menu showing '-- select your country --'), and '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:



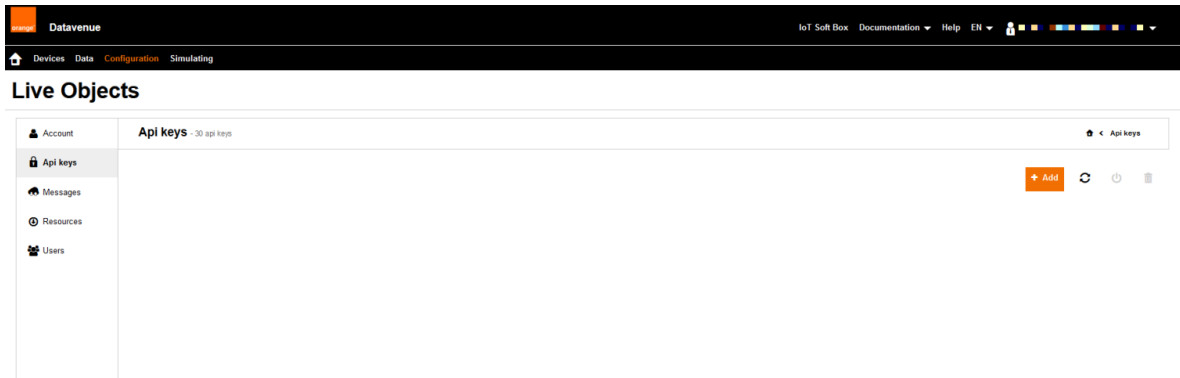
1. Fill the “Log in” form with your credentials:
 - your email address,
 - the password set during the activation phase,
2. Then click on the “Log in” button.

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.

The screenshot displays the 'Add an api key' form. It is divided into two main sections: 'Information' and 'Validity'. The 'Information' section includes fields for 'Name' (with the value 'Getting starter'), 'Description' (with the value 'Child of MasterKey'), and 'Roles' (with the value 'APIKEY_ADMIN'). The 'Validity' section includes fields for 'From' (with the value '11/17/2016 10:42 AM'), 'To' (with the placeholder 'enter an end date'), and 'TTL' (with the placeholder 'enter a time-to-live in seconds'). The form also features an '+ Add' button and a 'Back' button.

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 Platform](#), using:
 - an optional secure connection (TLS)
 - the Live Objects mode: [Json+Device](#)
- Publishes
 - 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.

Note:

Live Objects features (status, configuration, collected data, commands, resources) can be implemented or not according to the characteristics of your board (mainly the RAM size).

		<i>LinkIt One</i>
Hardware	Flash	16 MB
	RAM	4 MB
Live Objects Feature	Status	Yes
	Configuration	Yes
	Collected Data	Yes
	Commands	Yes
	Resources	Yes
Security	TLS	Yes

3.3.2. Packages dependences

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

1) *Mbed-tls, version 2.1.6*

<https://github.com/ARMmbed/mbedtls.git/#c60c31226fa746a363acb5a92ebb5d61d0d9de06>

Modified files:

- platform.h
- config.h

Added files:

- _config_all_disabled.h
- _config_mdk.h
- _config_mdk_min.h
- _config_samd.h
- _entropy_poll_alt.c

2) *Paho MQTT embedded-c (MQTTPacket and MQTTClient)*

<https://github.com/eclipse/paho.mqtt.embedded-c/#02323e1093f0414c1adcf03559e03a890b5f3a84>

3) *jsmn*

<https://github.com/zserge/jsmn/#1682c32e9ae5990ddd0f0e907270a0f6dde5cbe9>

4) *LiveObjects-iotSoftbox-mqtt-core (the Delivery1.0 release of library on github)*

<https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-core.git/#b18c7e1e0d7d2281848e22acbe45aa45c7d8d9af>

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™ ONE as a target.

1. Install the Arduino IDE (tested version: 1.8.1)

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 **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_params.h* to set your Live Objects Tenant API key.
- *Liveobjects_dev_config.h* , optionally to change some parameters to tune the iotsoftbox library.

- Open example sketch **File -> Examples -> LiveObjects iotsoftbox library -> liveobjects_sample_basic_mdk**
 - For LinkIt One board, 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**

3.3.4. Build

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

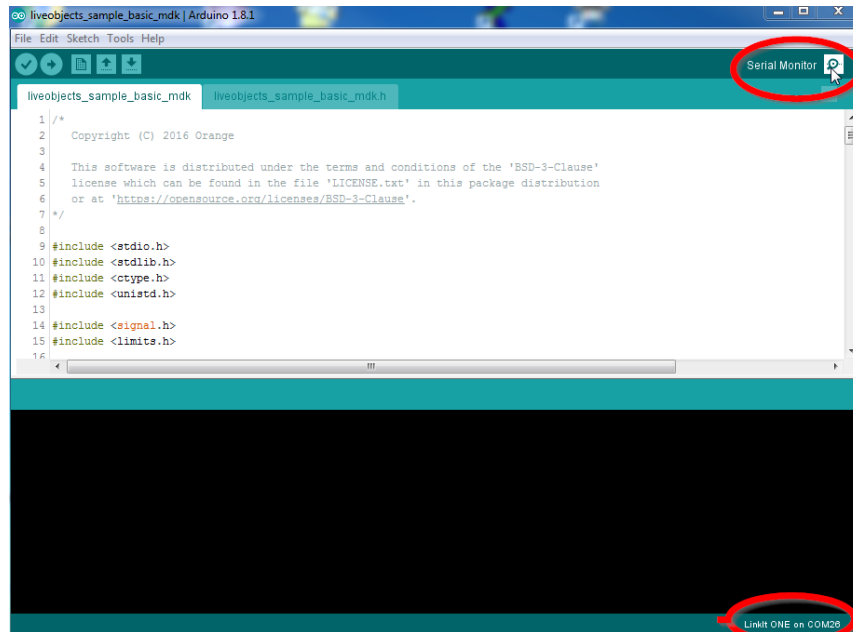
3.3.5. Launch

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

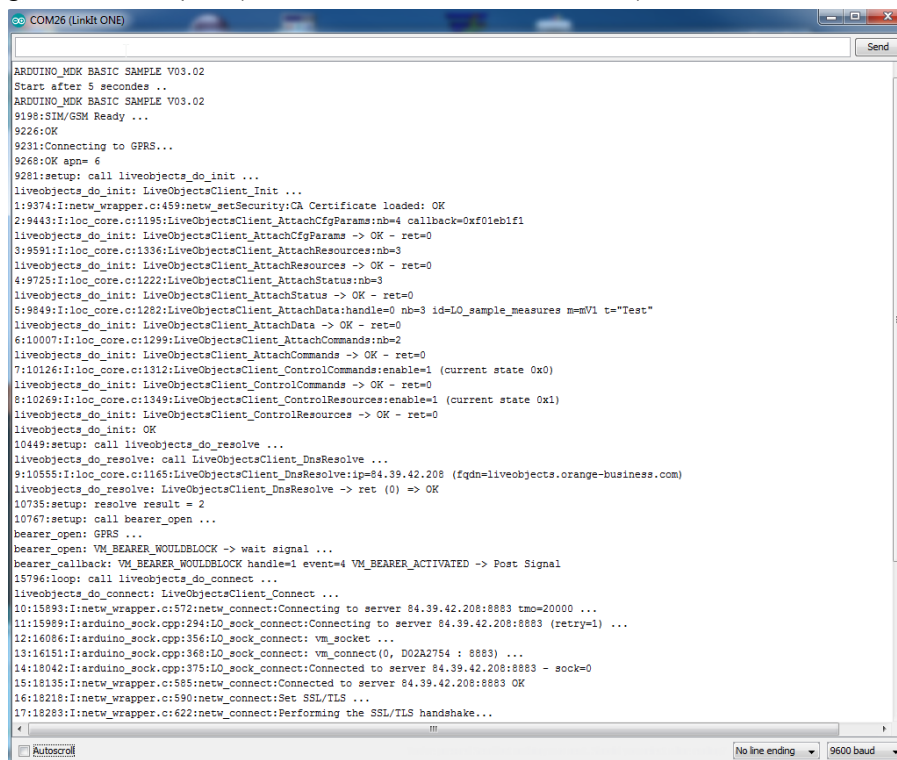
- Connect your board to your computer USB port running the Arduino IDE.
- Check that the correct board is chosen in **Tools -> Boards**.
- Verify that the IDE is using the correct COM port (**Tools -> Port**).
- To upload a program to your board: **Sketch -> Upload**.

3.3.5.2. Excute the application

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



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



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.
- If TLS is enabled.
 - Public Root Certificate
 - Certificate Common Name 'm2m.orange.com'

Therefore the user has only to define in the main file:

- Tenant *ApiKey* parameter

- if the TLS is enabled (value : 1) or not (value : 0)

```

/** Here, set your LiveObject ApiKey. 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_P2 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

```

To enable security :

You just have to set the SECURITY_ENABLED flag to 1 in `liveobjects_dev_param.h`

When TLS is enabled, security parameters must be defined/updated in the following header file `liveobjects_dev_security.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 LiveObjects 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 user header file `liveobjects_dev_params.h`:



```
/* Default LiveObjects device settings : name space and device identifier*/
#define LOC_CLIENT_DEV_NAME_SPACE      "LiveObjectsDomain"

#define LOC_CLIENT_DEV_ID              "LO_softboxlinux_01"
```

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):

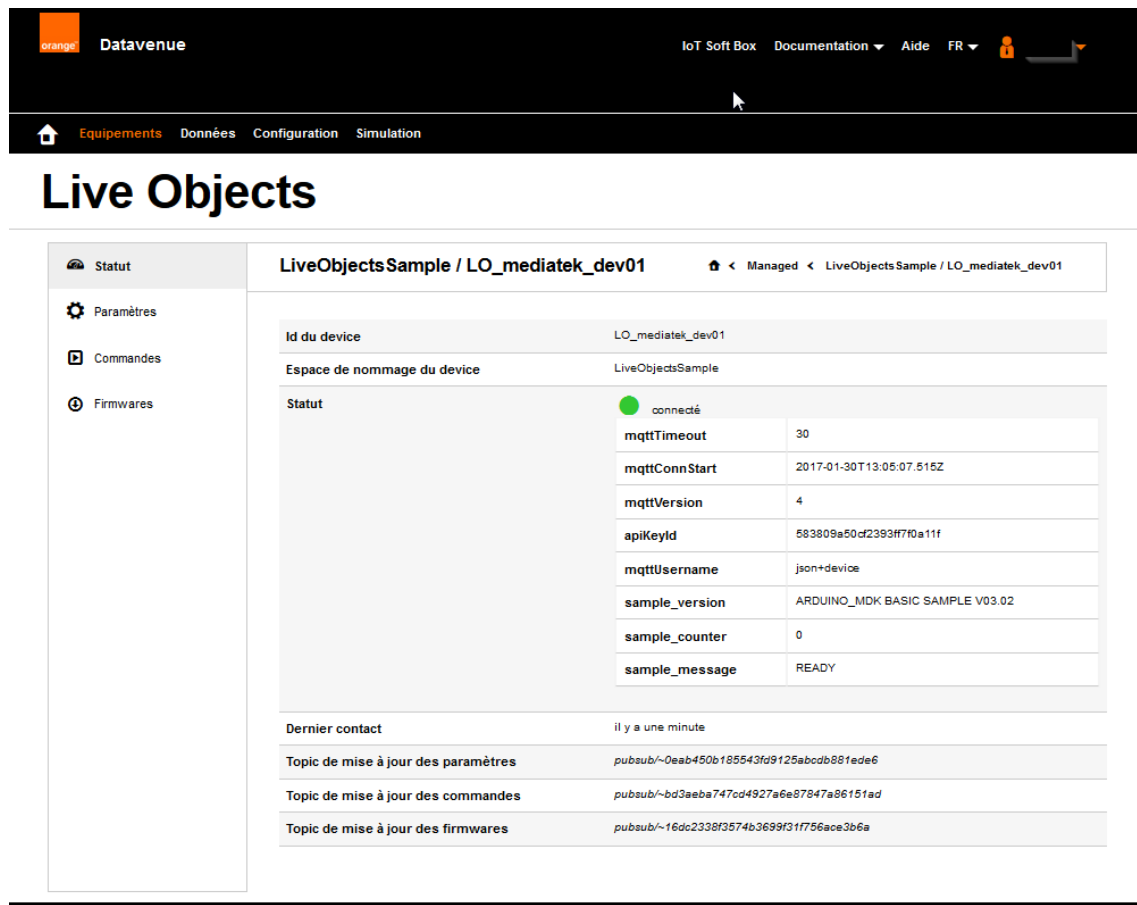
```
ret = LiveObjectsClient_PushStatus(appv_hdl_status);
```

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:



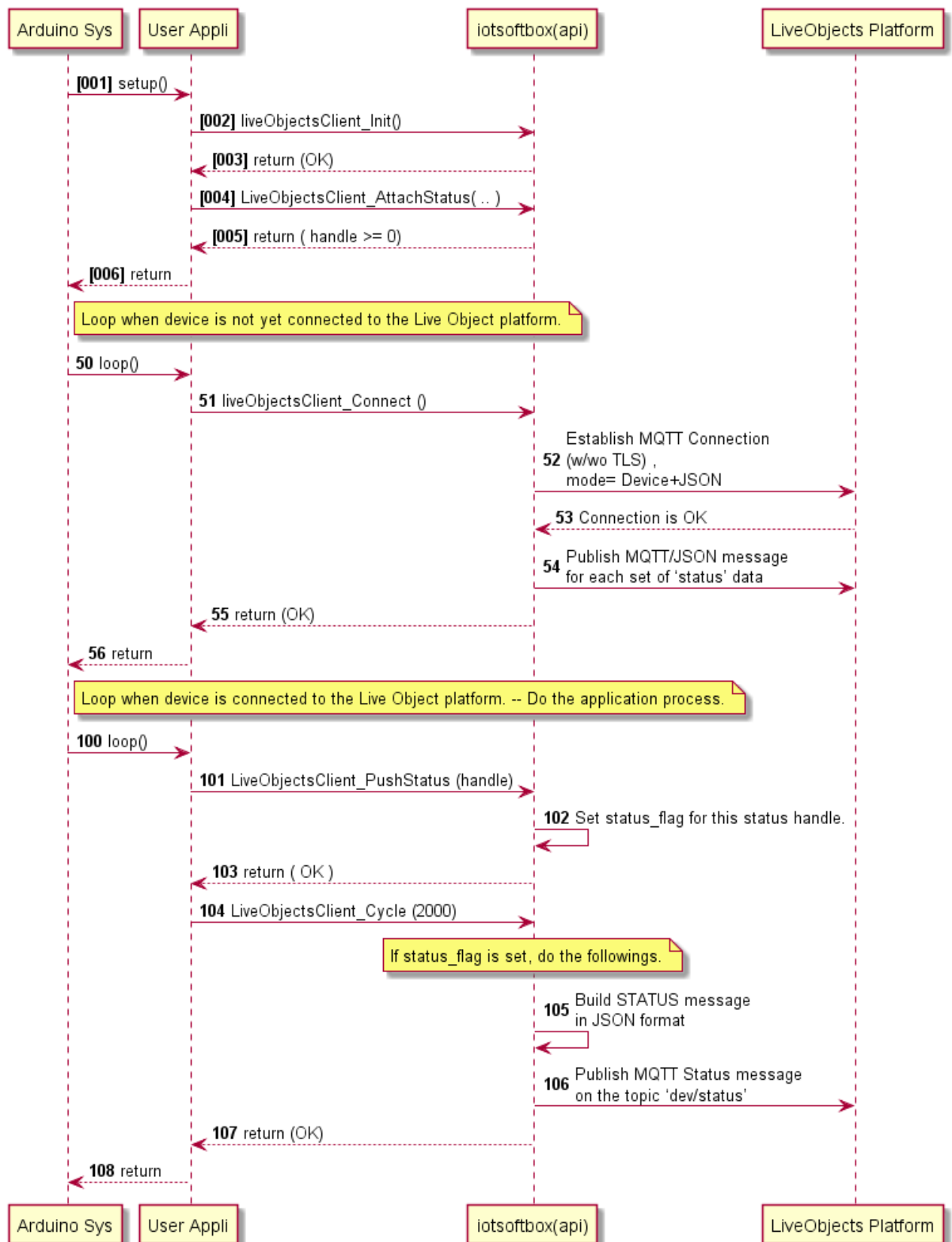
The screenshot shows the Datavenue Live Objects portal interface. The top navigation bar includes the Datavenue logo, links for IoT Soft Box, Documentation, Aide, and FR. The main navigation bar has links for Equipements, Données, Configuration, and Simulation. The page title is "Live Objects".

The main content area displays the status of a device named "LiveObjectsSample / LO_mediatek_dev01". The device is shown as "connecté" (connected) with a green status indicator. The status table lists the following information:

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	583809a50cd2393ff7f0a11f
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/~0aab450b185543fd9125abfdb881ede6
Topic de mise à jour des commandes	pubsub/~bd3aeba747cd4927a6e87847a86151ad
Topic de mise à jour des firmwares	pubsub/~16dc2338f3574b3699f31f756ace3b6a



4.5.4. Sequence Diagram



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. `paramUpdateCb`) 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->param_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.

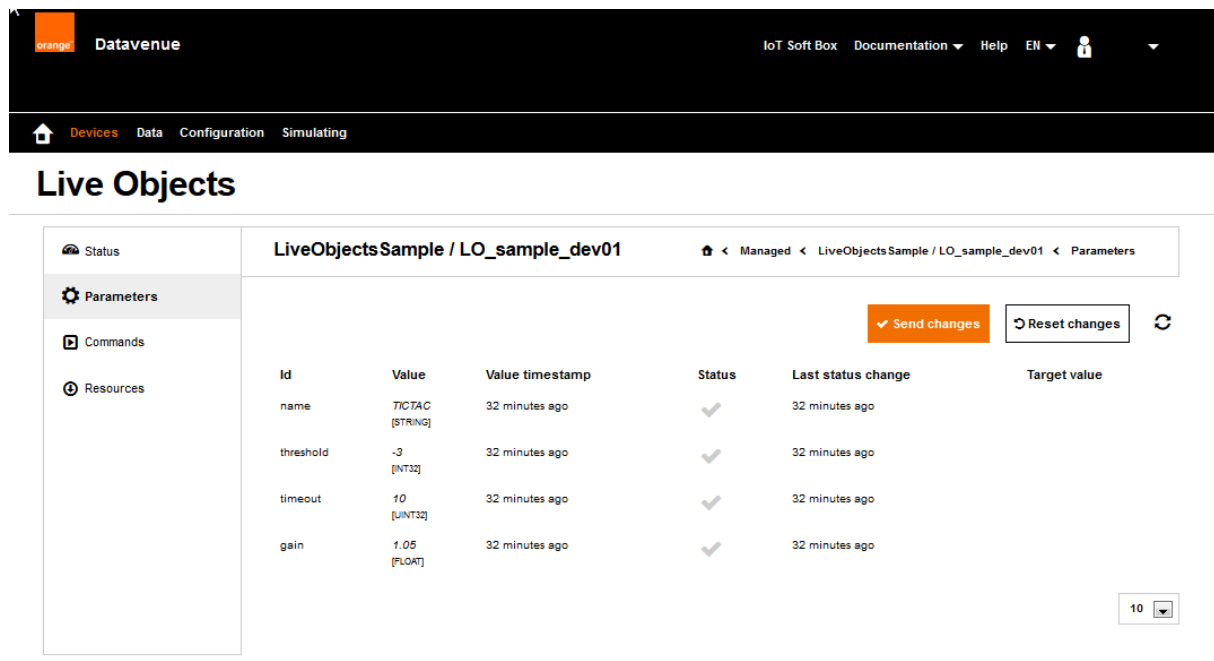
4.6.2. Push a set of configuration parameters

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

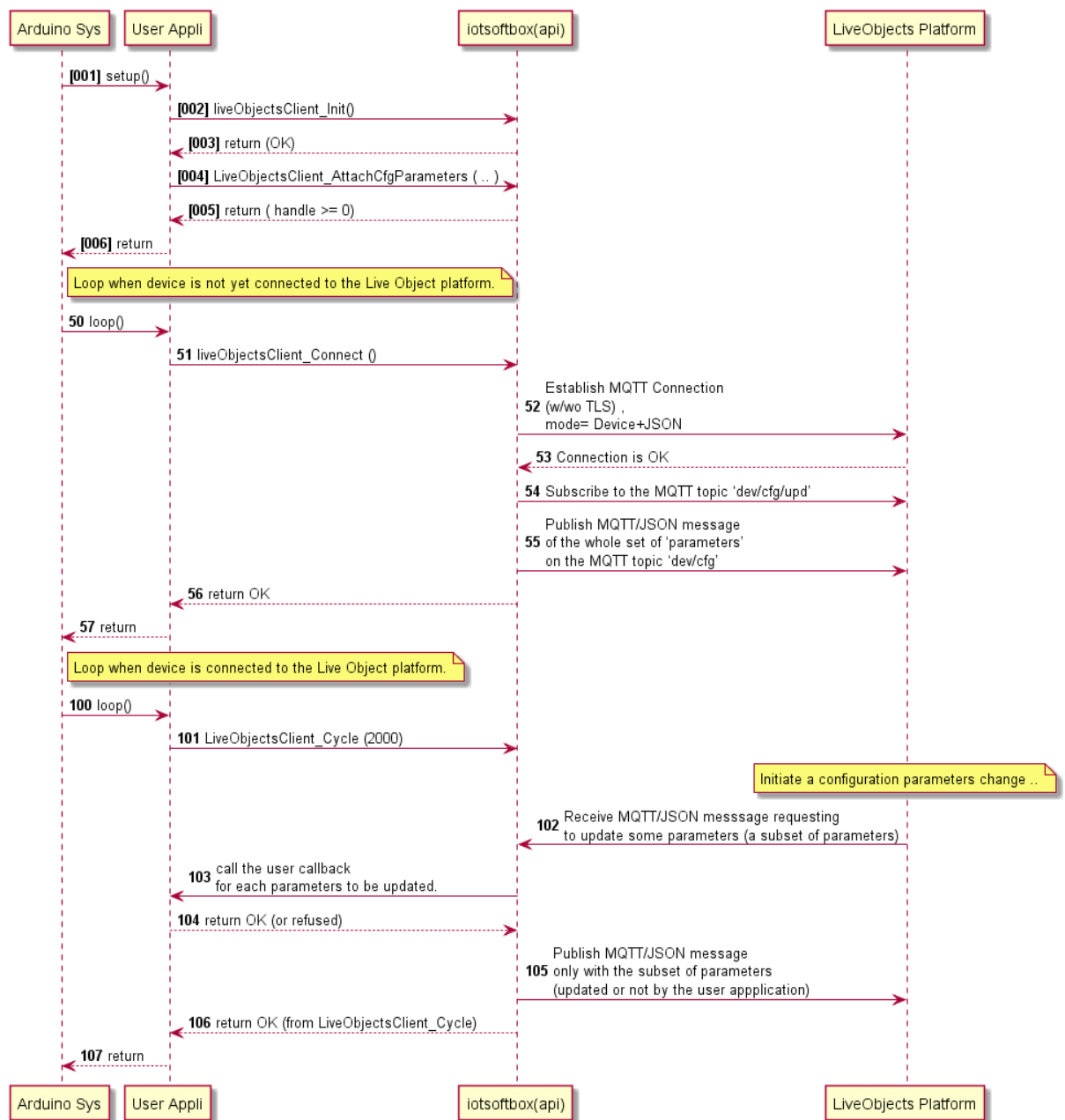
On the Datavenue Live Objects portal, the user can check the 'Parameters' of its connected device, but also change these initial values:



The screenshot shows the Datavenue Live Objects portal interface. The top navigation bar includes the Datavenue logo and links for IoT Soft Box, Documentation, Help, EN, and a user profile. Below this is a secondary navigation bar with links for Devices, Data, Configuration, and Simulating. The main content area is titled 'Live Objects' and displays the parameters for a device named 'LiveObjectsSample / LO_sample_dev01'. The parameters are listed in a table with columns for Id, Value, Value timestamp, Status, Last status change, and Target value. The parameters shown are name, threshold, timeout, and gain. The 'Send changes' button is highlighted in orange, and the 'Reset changes' button is in white. A pagination control shows '10' items per page.

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



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))
```

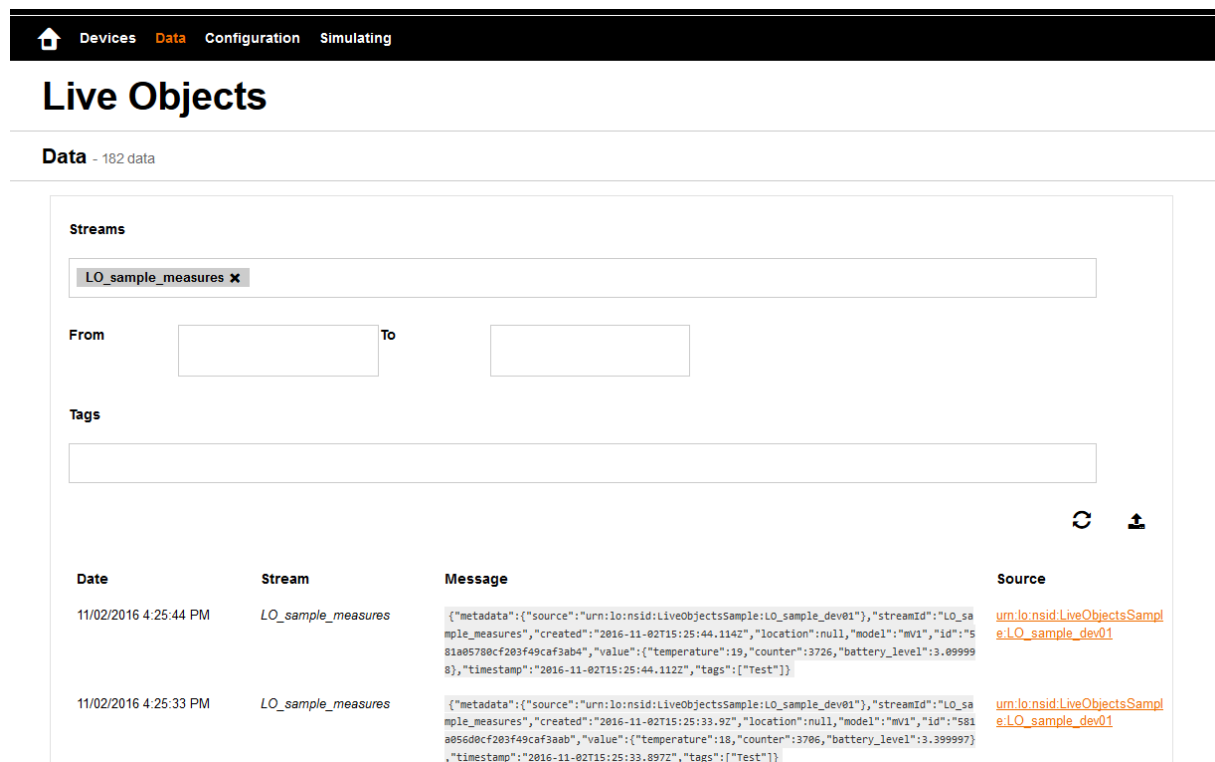
4.7.2. Push the set of collected data

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

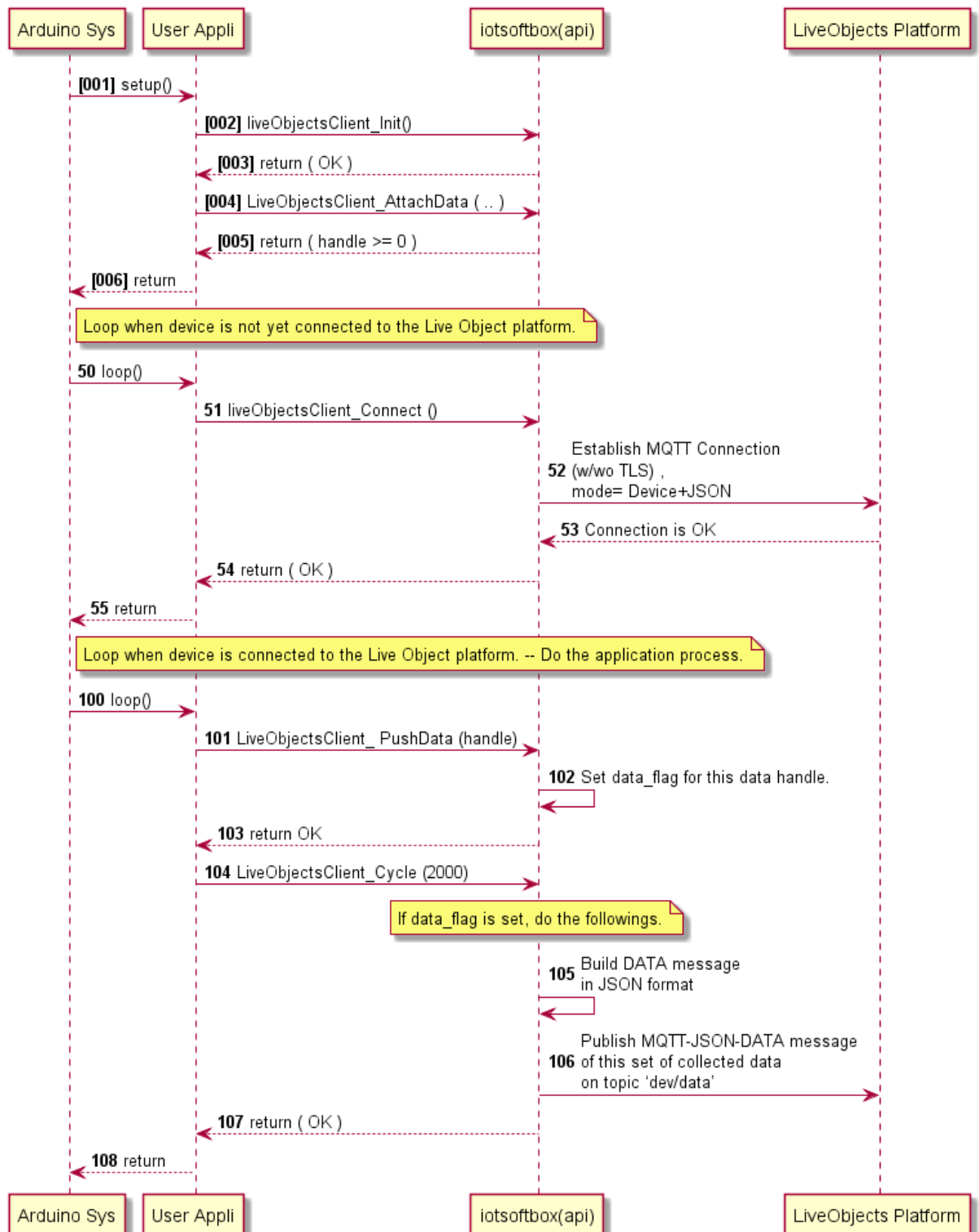
On the Datavenue Live Objects portal, the user can check the 'Collected Data' published by its connected device (here, filter is set to get only stream = LO_sample_measures):



The screenshot shows the 'Live Objects' portal interface. At the top, there's a navigation bar with 'Devices', 'Data', 'Configuration', and 'Simulating'. Below this, the 'Live Objects' title is displayed. Under 'Data - 182 data', there's a 'Streams' section with a filter 'LO_sample_measures'. Below the filter, there are input fields for 'From' and 'To' dates, and a 'Tags' input field. A table of data is shown below, with columns for Date, Stream, Message, and Source. The table contains two entries for the stream 'LO_sample_measures' from 11/02/2016 4:25:44 PM and 11/02/2016 4:25:33 PM. The messages are JSON objects containing metadata and sensor data like temperature, counter, and battery level.

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": "2016-11-02T15:25:33.92Z", "location": null, "model": "mv1", "id": "581a05600cf203f49caf3ab4", "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



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

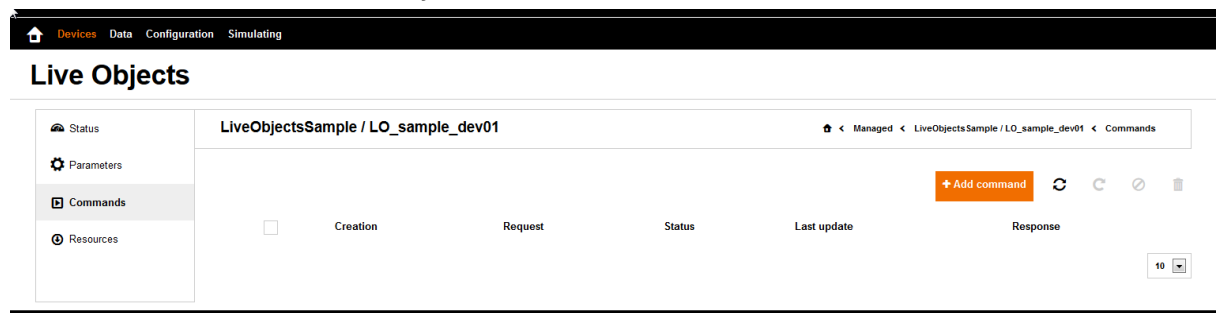
As soon as the device is ready (or not) to process commands, the application can enable (or disable) the 'command' feature by calling the function:

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

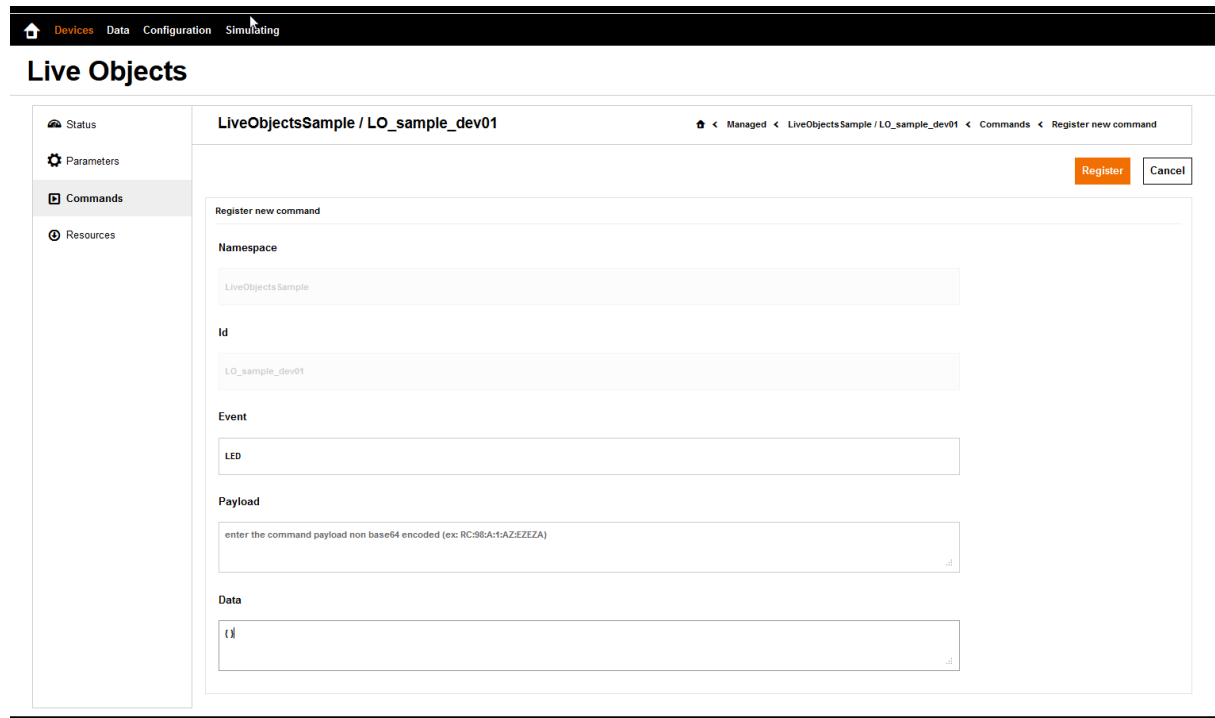
4.8.3. Use of Live Objects Portal to send a command

On the Live Objects Portal,

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



- Click on button '+ Add command'

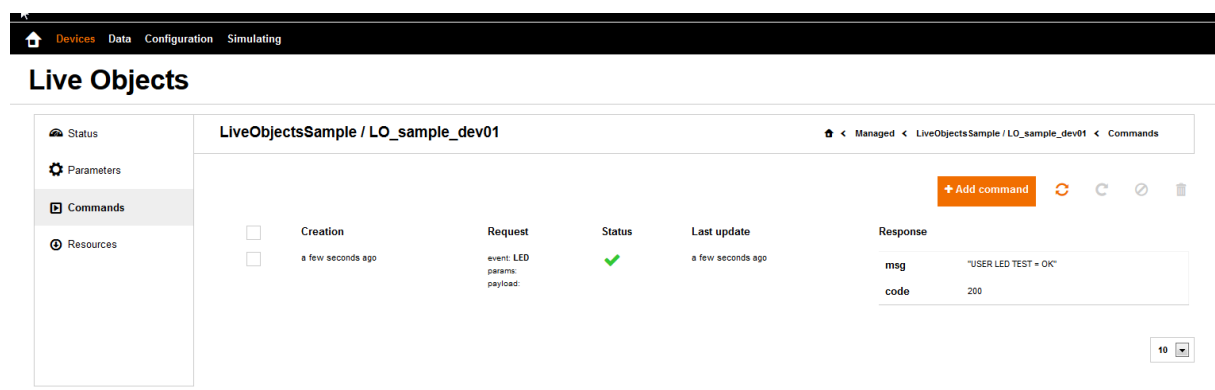


The screenshot shows the 'Live Objects' web interface. The top navigation bar includes 'Devices', 'Data', 'Configuration', and 'Simulating'. The main header is 'Live Objects'. On the left, there is a sidebar with 'Status', 'Parameters', 'Commands', and 'Resources'. The main content area is titled 'LiveObjectsSample / LO_sample_dev01'. It features a 'Register new command' form with the following fields:

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

At the top right of the form, there are 'Register' and 'Cancel' buttons.

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

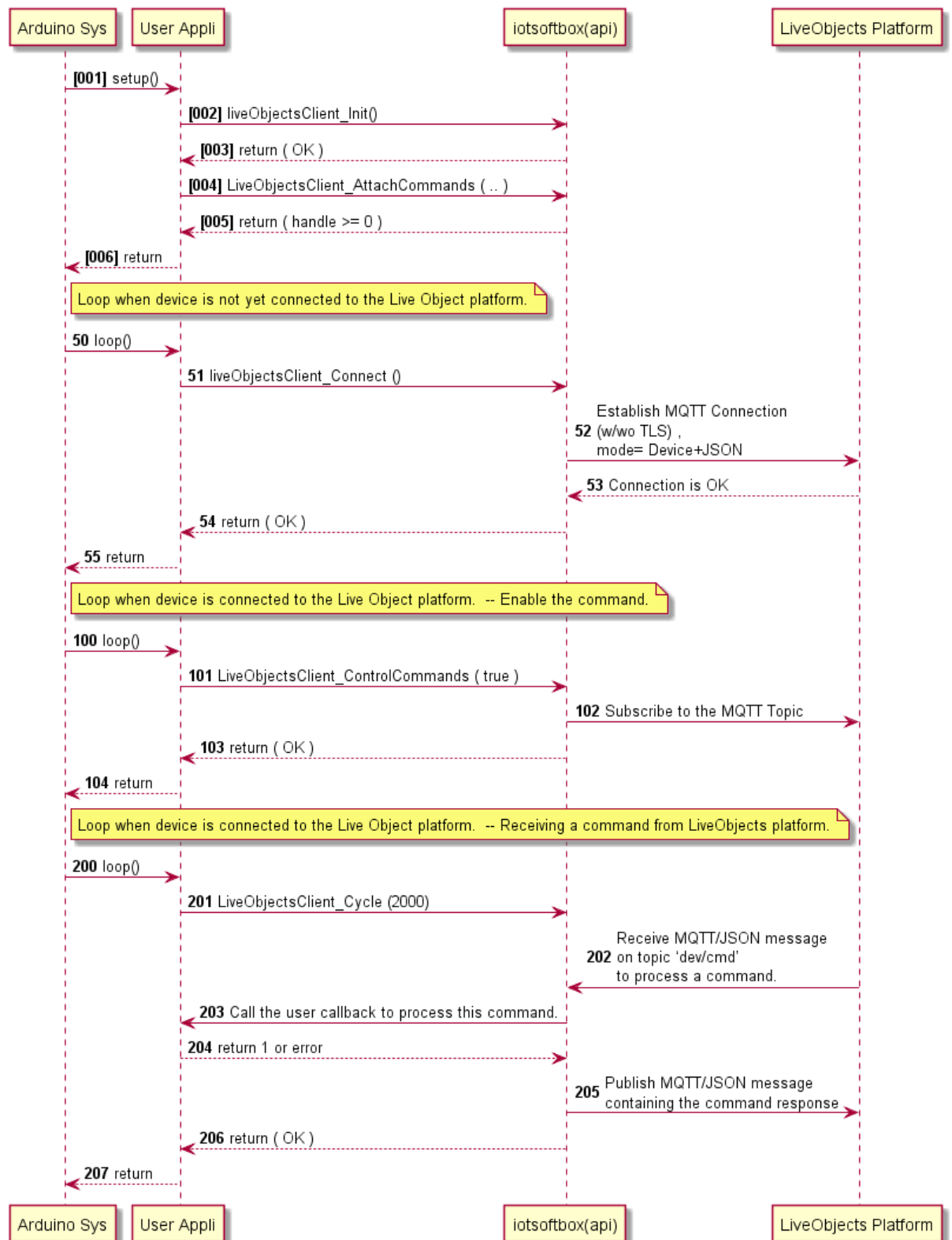


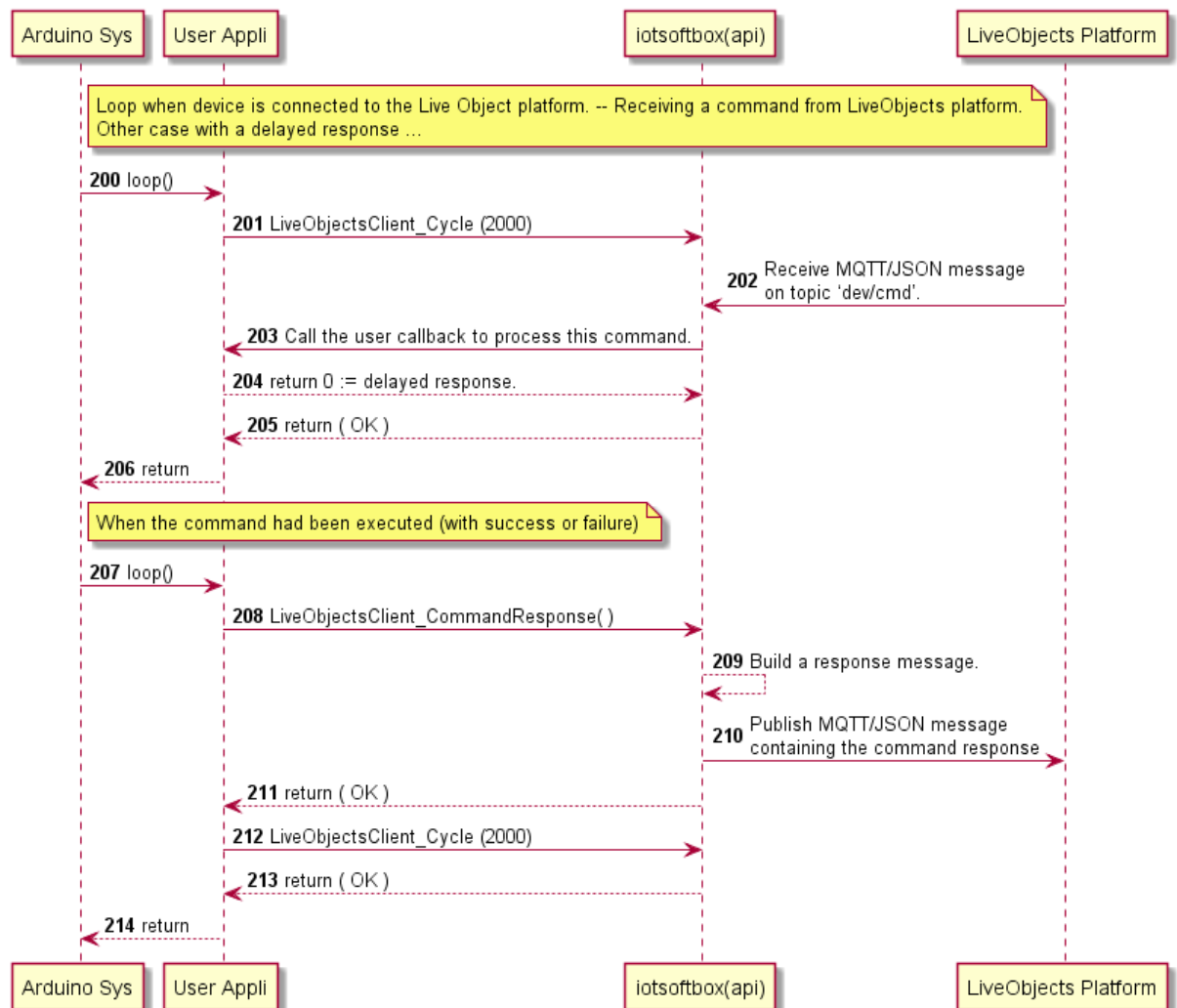
The screenshot shows the 'Live Objects' web interface after registering a command. The main content area is titled 'LiveObjectsSample / LO_sample_dev01'. It features a table with the following columns: 'Creation', 'Request', 'Status', 'Last update', and 'Response'. The table contains one row of data:

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

At the top right of the table, there is an 'Add command' button and several icons (refresh, undo, redo, delete). At the bottom right, there is a pagination control showing '10' items.

4.8.4. Sequence Diagram





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_RscGetChunk(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 several time if you're using a very low capacity device. Once you have the data you can process them.

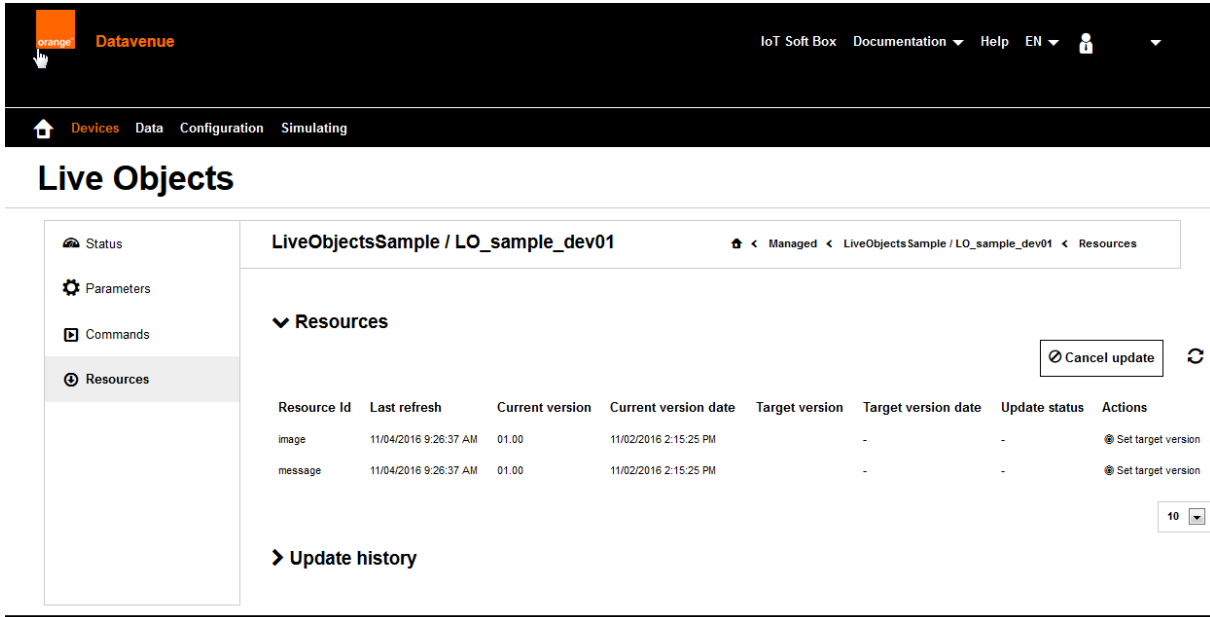
4.9.2. Enable/disable 'resources' feature

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.



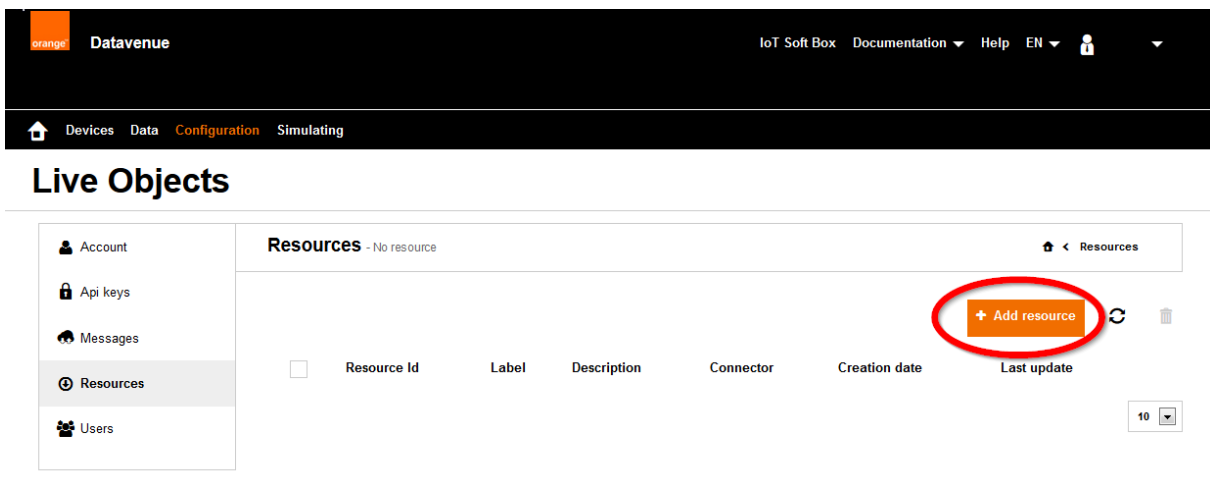
The screenshot shows the Datavue interface with the 'Live Objects' section selected. The breadcrumb trail indicates the path: Managed < LiveObjectsSample / LO_sample_dev01 < Resources. The 'Resources' tab is active, displaying a table of resources for the device 'LiveObjectsSample / LO_sample_dev01'.

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

Additional UI elements include a 'Cancel update' button, a refresh icon, and a pagination control showing '10' items.

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

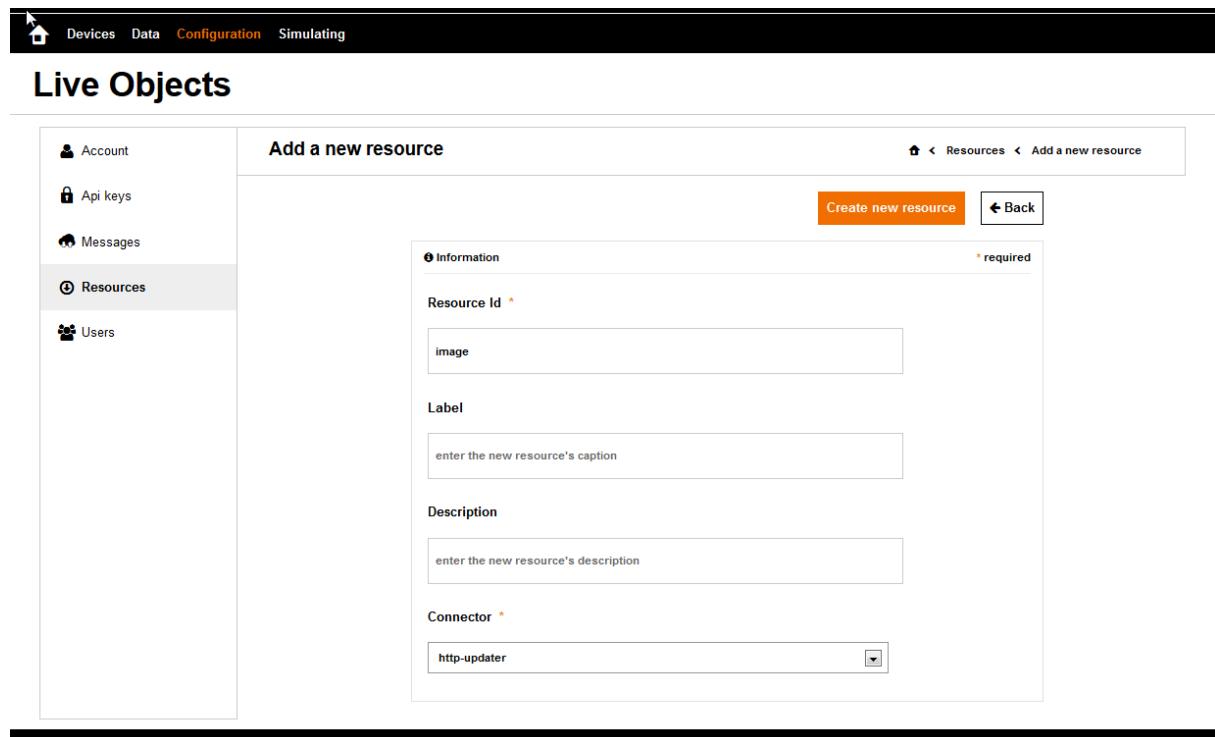
Now, the user can create a new resource on the Live Objects platform, in the tab 'Configuration->Resources', associated to these resources



The screenshot shows the Datavue interface with the 'Configuration' tab selected. The breadcrumb trail indicates the path: Resources < Resources. The 'Resources' tab is active, displaying a table with columns: Resource Id, Label, Description, Connector, Creation date, and Last update. A red circle highlights the '+ Add resource' button in the top right corner of the table area.

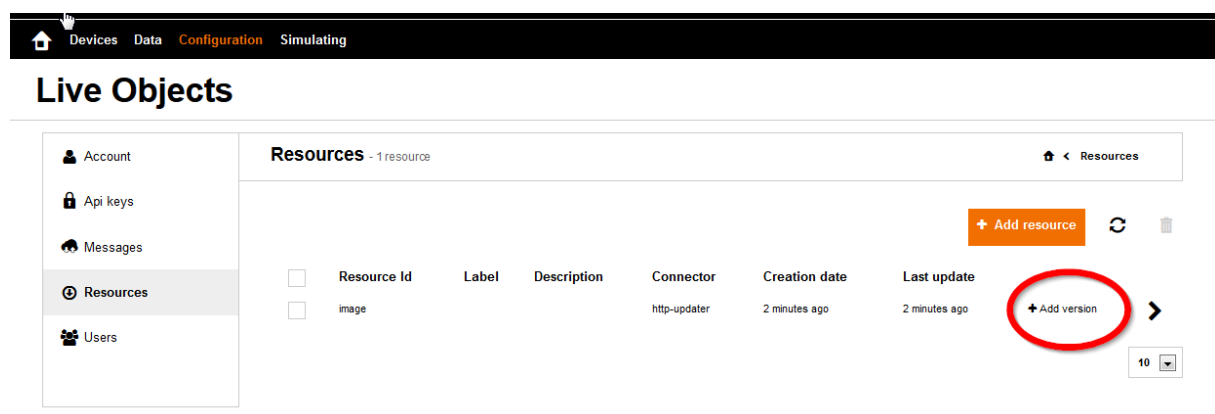
Two fields are mandatory

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



The screenshot shows the 'Live Objects' interface with the 'Add a new resource' form. The left sidebar contains links to Account, Api keys, Messages, Resources (selected), and Users. The top navigation bar includes Devices, Data, Configuration (selected), and Simulating. The form has a title 'Add a new resource' and a breadcrumb 'Resources < Add a new resource'. It features a 'Create new resource' button and a 'Back' button. The form is divided into an 'Information' section (marked as required) with fields for 'Resource Id' (containing 'image'), 'Label' (with placeholder 'enter the new resource's caption'), 'Description' (with placeholder 'enter the new resource's description'), and 'Connector' (a dropdown menu showing 'http-updater').

The result is the following:

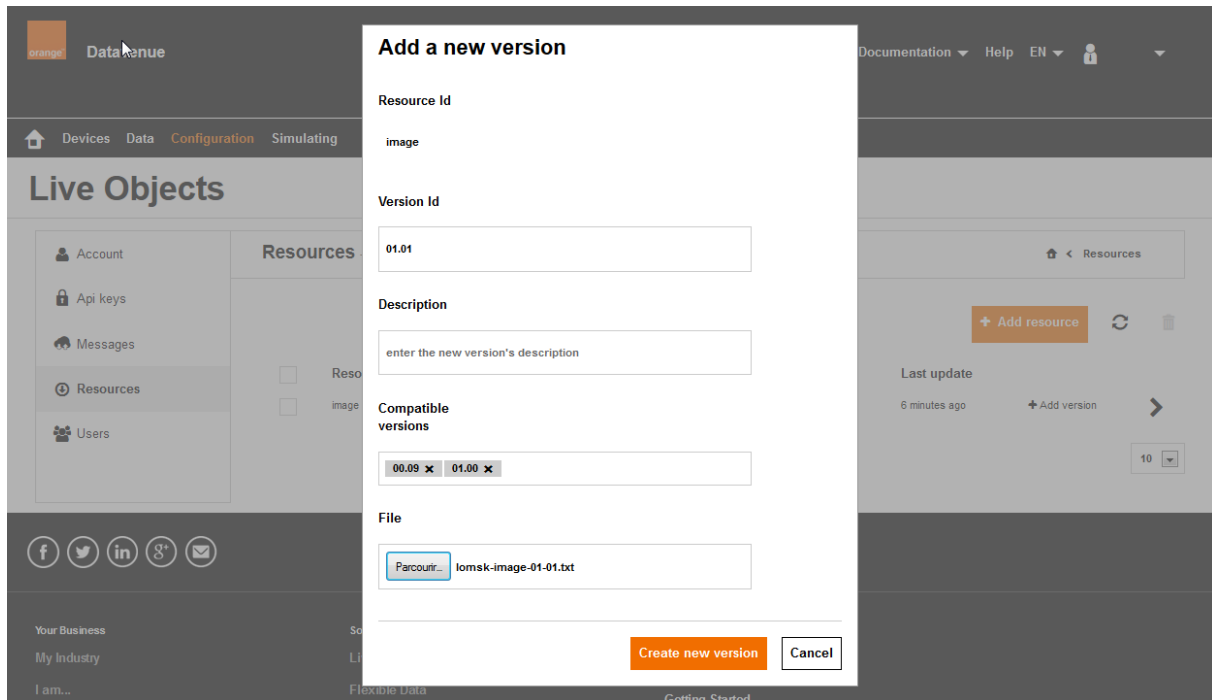


The screenshot shows the 'Live Objects' interface with the 'Resources' table. The left sidebar is the same as the previous screenshot. The top navigation bar is the same. The table title is 'Resources - 1 resource' with a breadcrumb 'Resources'. The table has columns: Resource Id, Label, Description, Connector, Creation date, and Last update. There is one resource with 'Resource Id' 'image' and 'Connector' 'http-updater'. The 'Creation date' and 'Last update' are both '2 minutes ago'. To the right of the table, there is an 'Add resource' button, a refresh icon, a trash icon, and a '+ Add version' button (circled in red). A pagination control shows '10' and a dropdown arrow.

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

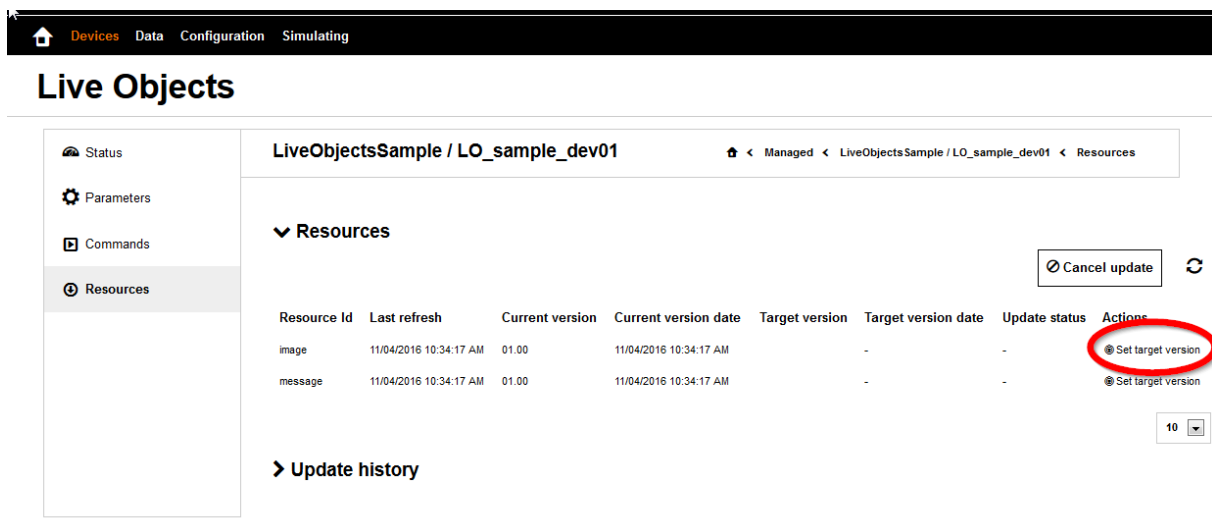
Then, a new resource version can be attached to this resource 'image', by specifying:

- **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

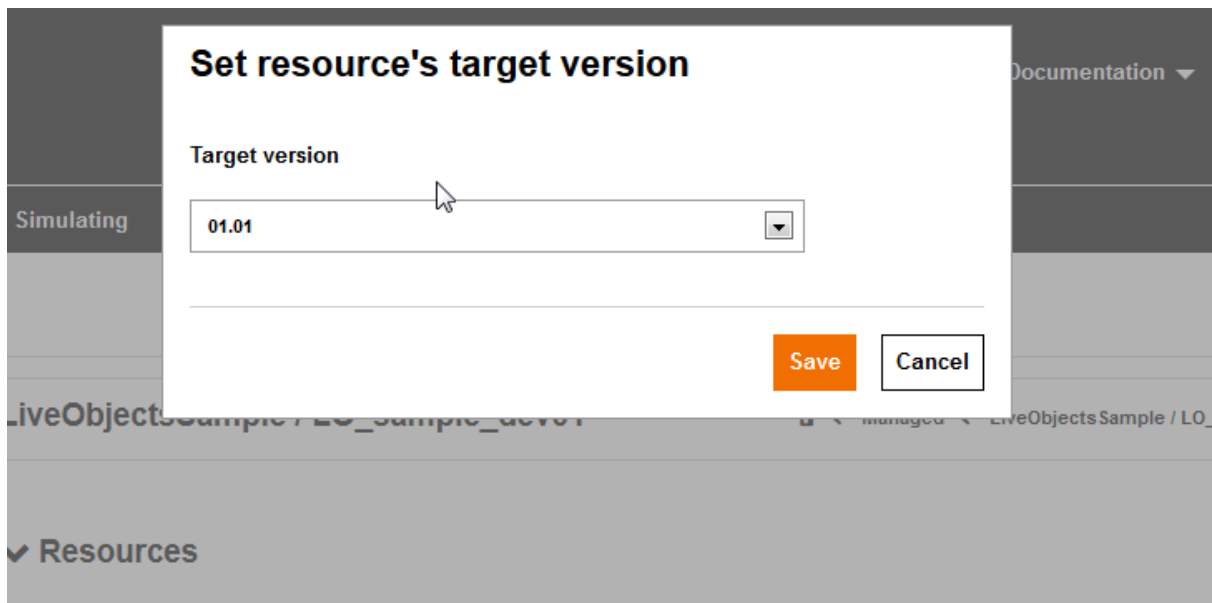


Now, the resource update request can be launched for this device:

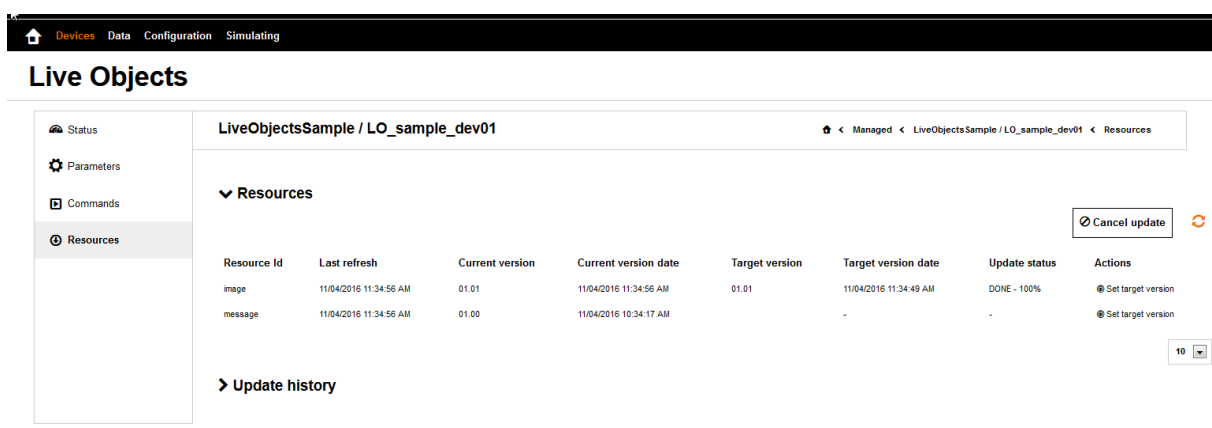
- Go to the 'devices' tab.
- Select your connected device, here it is "LiveObjectsSample / LO_sample_dev01"
- Go to the 'resources' tab



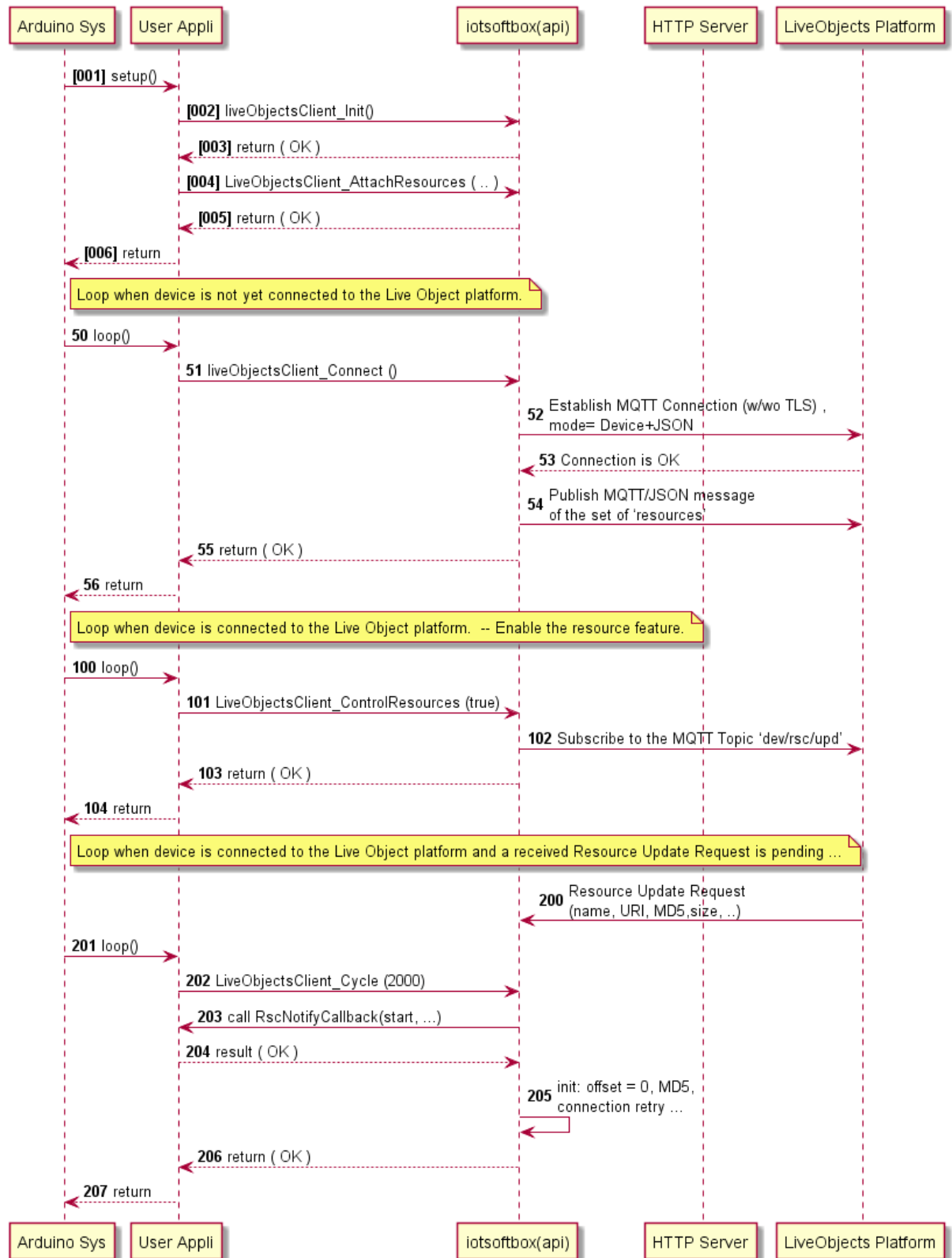
- Click on 'Set target version'
- And select the resource version to download on device



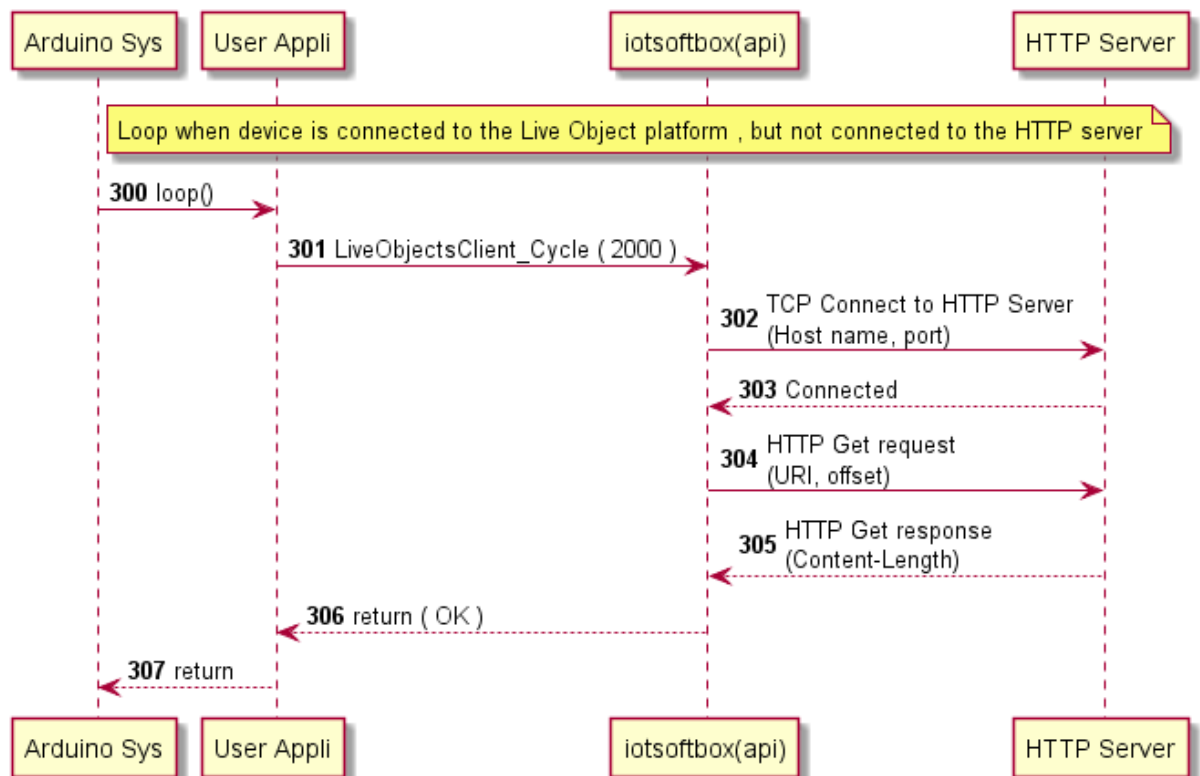
At the end of transfer, after refreshing the web page, the current version should be equal to the target version:



4.9.4. Sequence diagram



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



An example of URI is:

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

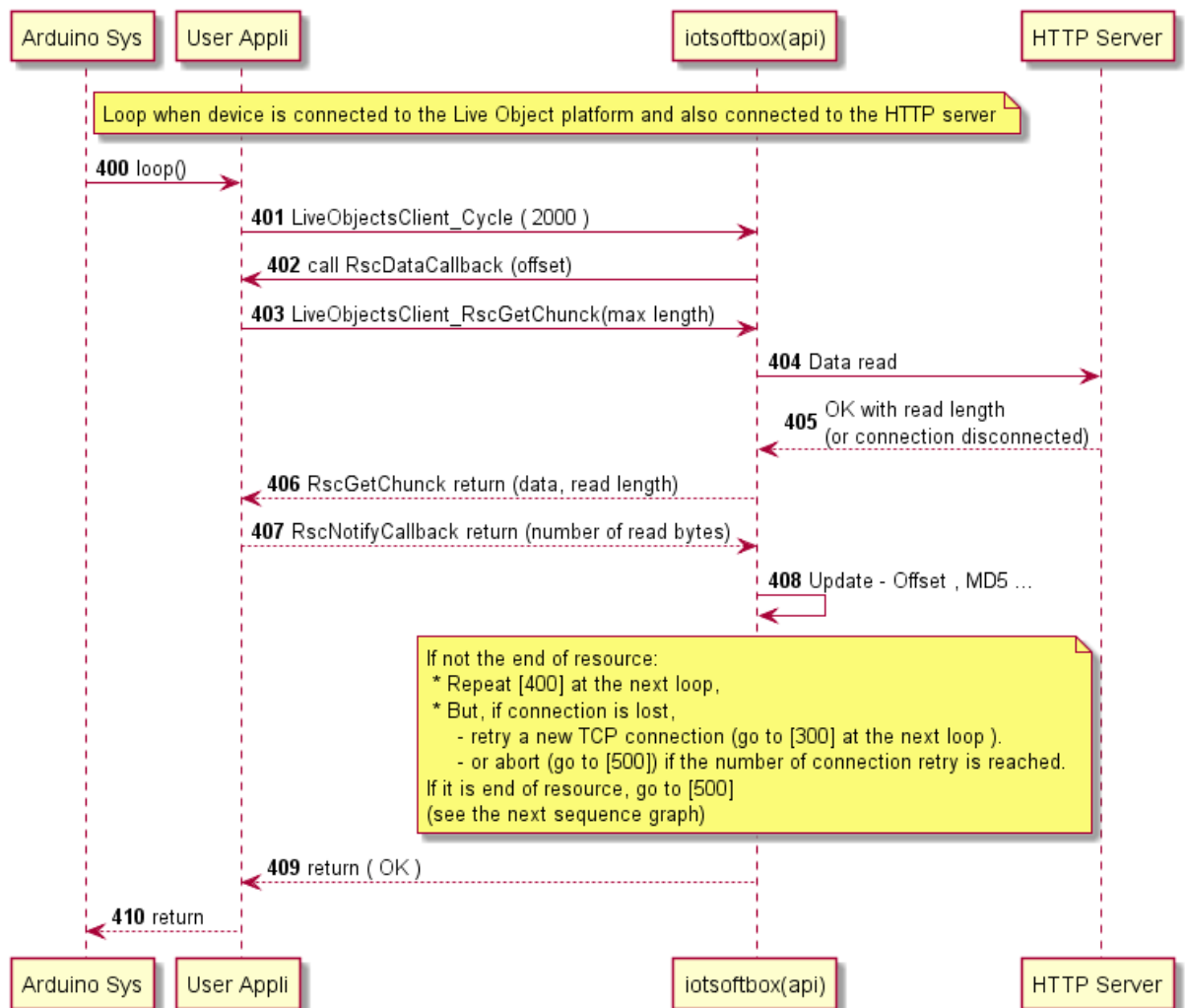
And the header of HTTP Get Response is similar to:

```

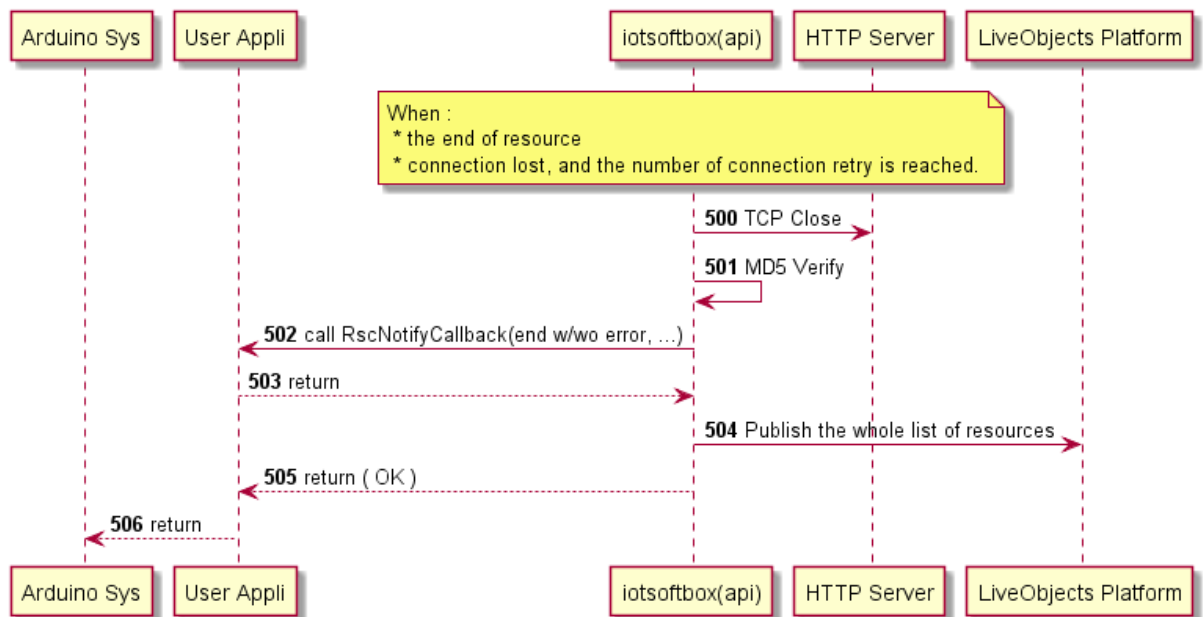
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Fri, 04 Nov 2016 10:34:50 GMT
Content-Type: application/force-download; charset=UTF-8
Content-Length: 1974
Connection: close
X-Application-Context: lo-http-updater:prod:8080
Access-Control-Allow-Headers: X-Requested-With, Content-Type
Access-Control-Allow-Credentials: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



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



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 their default values in the header file `liveobjects-client/LiveObjectsClient_Config.h`.

Then, the user application can overwrite these 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_MBEDTLS`
Implement or not the mbedtls feature
- `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
 - o No memory allocation
 - o No message queue
- `LOC_SERV_TIMEOUT`
Connection Timeout in milliseconds (default 20 seconds)
- `LOC_MQTT_API_KEEPLIVEINTERVAL_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 instantiated when the MQUEUE is not implemented.