



# IoT Soft Box Starter Kit

User Manual for `iotsoftbox-mqtt` library

Linux Edition

## Table of contents

<b>1. INTRODUCTION.....</b>	<b>4</b>
1.1. Document purpose .....	4
1.2. Reference documents .....	4
<b>2. OVERVIEW .....</b>	<b>5</b>
2.1. What is Live Objects?.....	5
2.2. Linux .....	5
2.3. IoT Soft Box .....	6
<b>3. GETTING STARTED.....</b>	<b>7</b>
3.1. Hardware Environment.....	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 .....	12
3.3.3. Configure workstation for cross compilation .....	13
3.3.4. Build .....	13
3.3.5. Launch .....	15
<b>4. DETAILED FEATURES .....</b>	<b>17</b>
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 .....	23
4.6. Parameters .....	23
4.6.1. Attach a set of configuration parameters .....	24
4.6.2. Push a set of configuration parameters.....	26
4.6.3. Use of Live Objects Portal to set/change parameters .....	26
4.6.4. Sequence Diagram .....	27
4.7. Collected Data .....	27
4.7.1. Attach a set of collected data .....	28
4.7.2. Push the set of collected data .....	28
4.7.3. Use of Live Objects Portal to view data stream .....	29
4.7.4. Sequence Diagram .....	30
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 .....	32
4.8.4.	Sequence Diagram .....	33
4.9.	Resources.....	34
4.9.1.	Attach a set of resources.....	34
4.9.2.	Enable/disable 'resources' feature.....	36
4.9.3.	Use of Live Objects Portal to create and update a resource .....	37
4.9.4.	Sequence diagram .....	41
<b>5.</b>	<b>ADDITIONAL INFORMATION.....</b>	<b>44</b>
5.1.	Doxygen documentation .....	44
5.2.	Debug.....	44
5.2.1.	Linux environment .....	44
5.3.	IoT Soft Box Library Configuration.....	45

# 1. Introduction

## 1.1. Document purpose

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

- Overview
- Getting started
- Detailed Features
- Additional Information

## 1.2. Reference documents

#	Origin	Title
1	Orange	<a href="#">Datavenue Live Objects - complete guide</a> (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, ressources, firmware, etc.),
- **Message Routing** between devices and business applications,
- **Data Management**: 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. Linux

Linux in embedded systems is already being used in consumer electronics like smartTVs or smartphones. In fact the Linux kernel has been ported to many CPU architectures like ARM or AVR32 for the most well-known.

- Linux is a mature and stable alternative to the proprietary OS.
- Linux supports a huge variety of applications and networking protocols.
- Linux is scalable so it can be used in small memory space; also, kernel footprint is less than 500 KB.
- Linux has attracted a huge number of active developers, enabling rapid support of new hardware architectures, platforms, and devices.
- Linux is largely accepted by hardware vendors, chip makers, single board computer maker etc.
- Linux being an open source operating system, it has a huge community supporting various projects to standardize build systems like [OpenWrt](#) or [Yocto](#).

**Live Objects iotsoftbox-mqtt library** is compatible with Linux platform.

## 2.3. IoT Soft Box

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

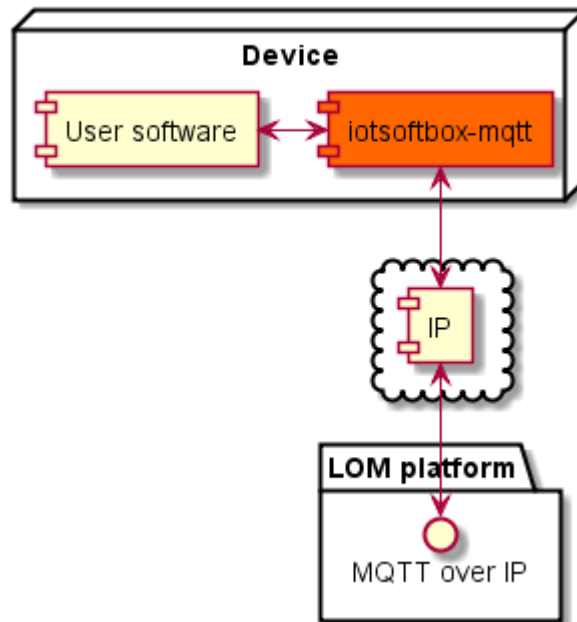


Figure 1 – IoT Soft Box integration in a system

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/o TLS.

The IoT Soft Box can run on devices connected to Internet through Ethernet, Wi-Fi, 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

To test our SDK, use a compatible hardware, like:

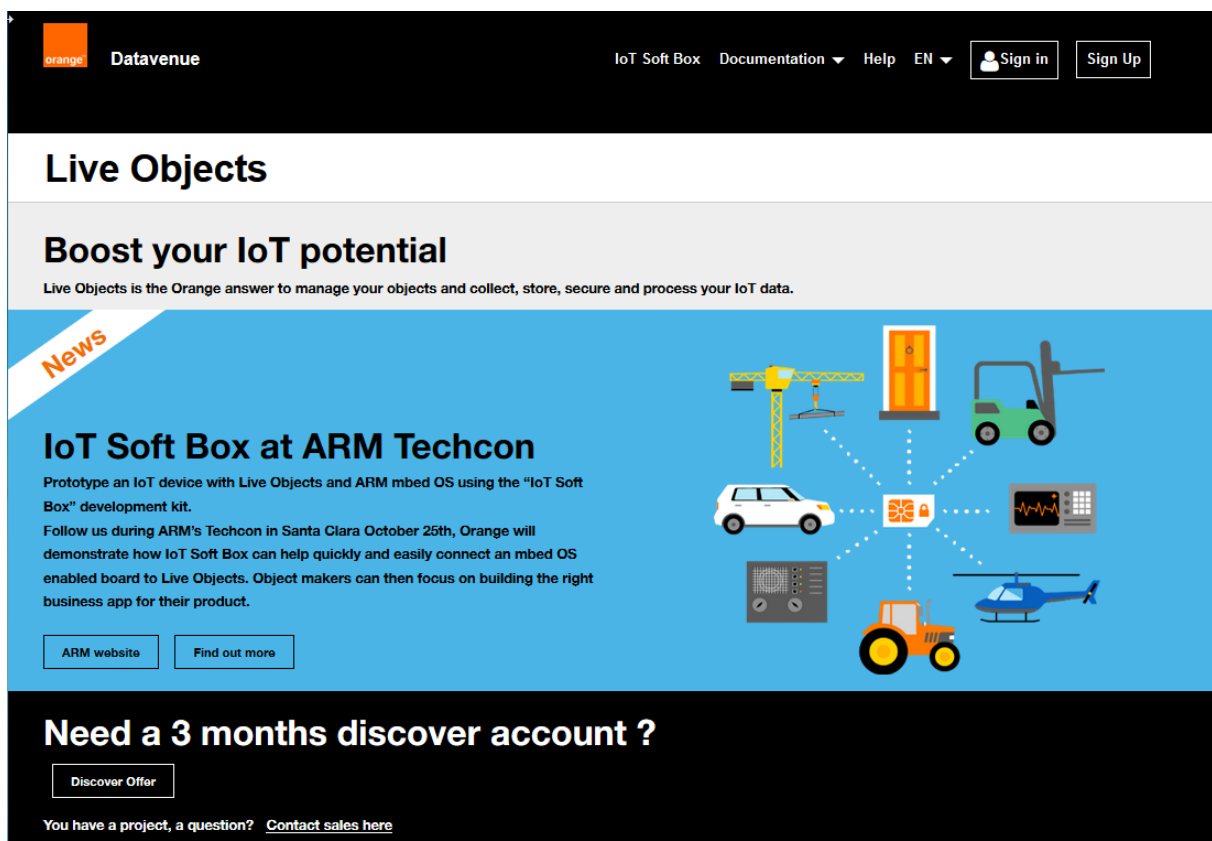
- A Raspberry Pi model B with [Raspbian Jessie](#) as Linux distribution
- [A Raspberry Pi 3 model B](#), with [Raspbian Jessie](#) as Linux distribution

### 3.2. Access to Live Objects

#### 3.2.1. Account creation

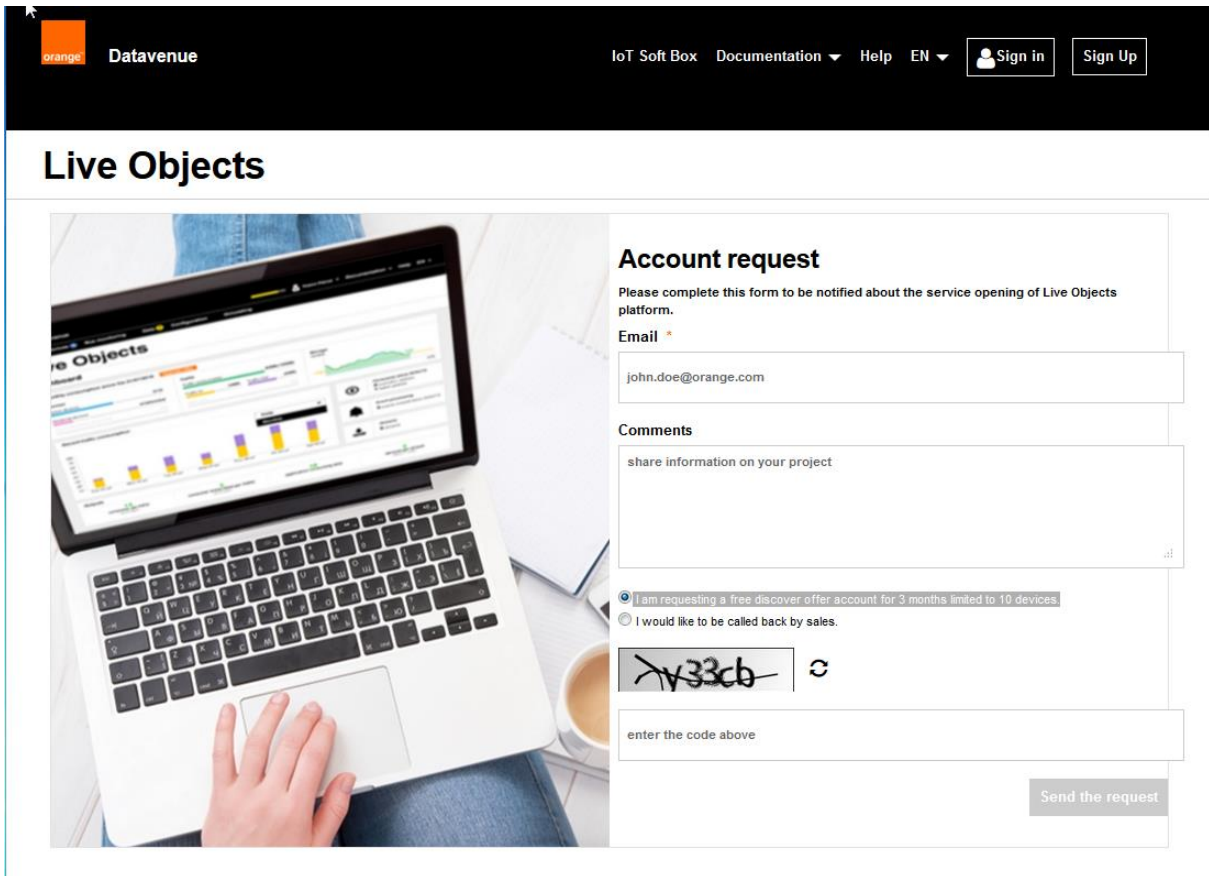
In order to use Live Objects, you need to have a dedicated account on the service.

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 section titled 'Live Objects' with the subtitle 'Boost your IoT potential'. A banner below this states 'Live Objects is the Orange answer to manage your objects and collect, store, secure and process your IoT data.' The main content area features a 'News' section titled 'IoT Soft Box at ARM Techcon' with text about prototyping IoT devices and a 'Find out more' button. To the right of the text is a diagram showing various IoT devices (car, crane, forklift, tractor, helicopter, server, and a central hub) connected by dotted lines. At the bottom, there is a section titled 'Need a 3 months discover account ?' with a 'Discover Offer' button and a link to 'Contact sales here'.

- 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'.



**Datavenue** IoT Soft Box Documentation Help EN Sign in Sign Up

## 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

- 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](#)

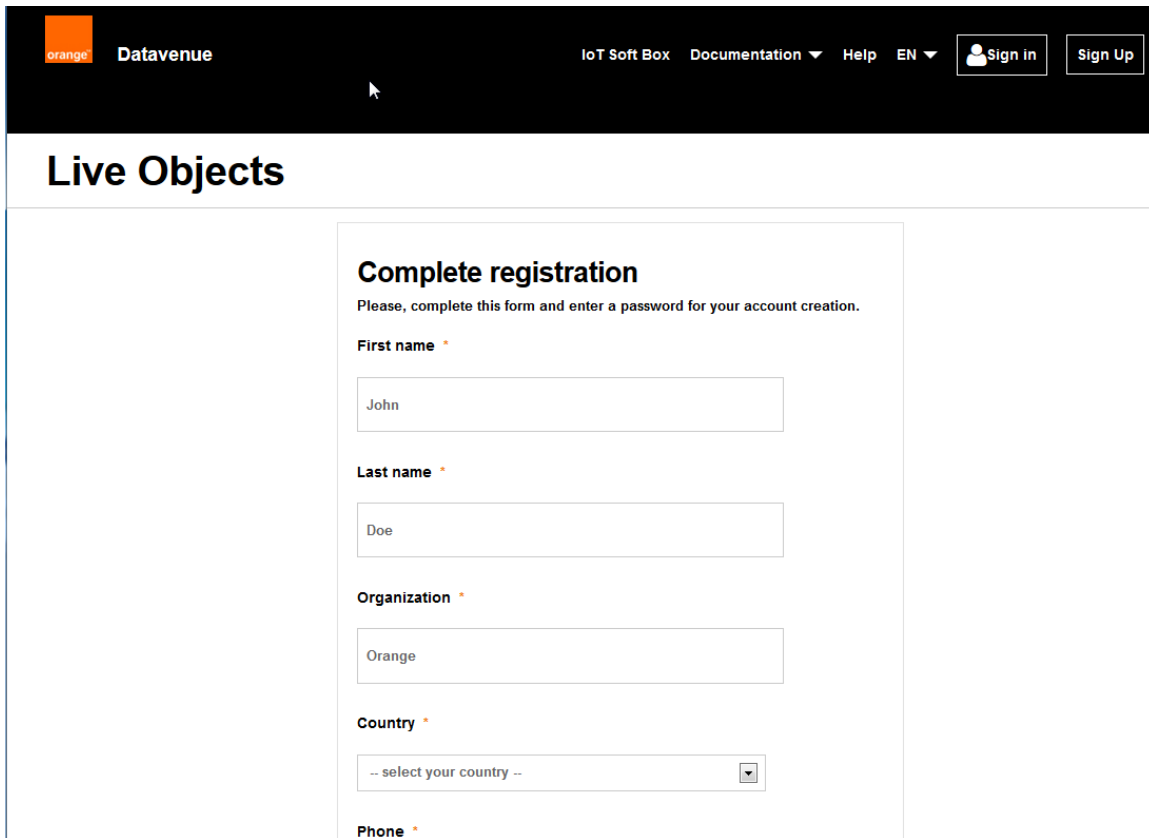
Thank you for your trust.

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 Datavenue logo on the left and navigation links (IoT Soft Box, Documentation, Help, EN) and login buttons (Sign in, Sign Up) on the right. Below the header, the page title "Live Objects" is displayed. The main content area features a "Complete registration" form. The form includes fields for First name (John), Last name (Doe), Organization (Orange), Country (a dropdown menu showing "-- select your country --"), and Phone. A red asterisk indicates required fields.

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

**First name \***

John

**Last name \***

Doe

**Organization \***

Orange

**Country \***

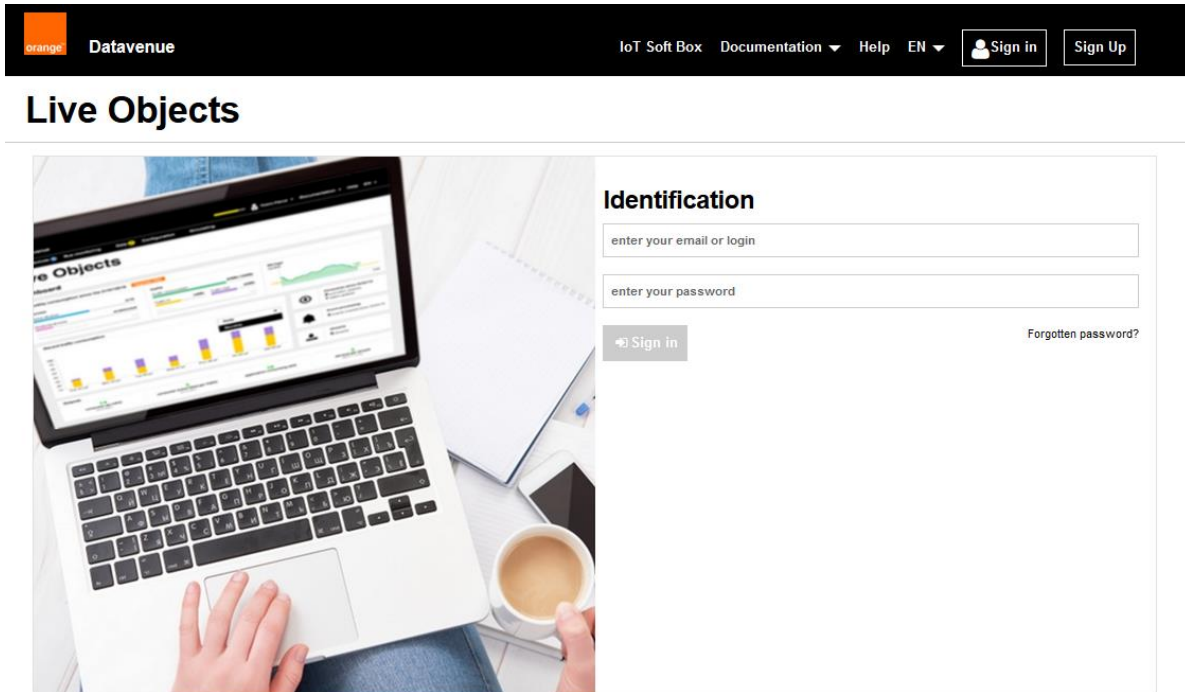
-- select your country --

**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.

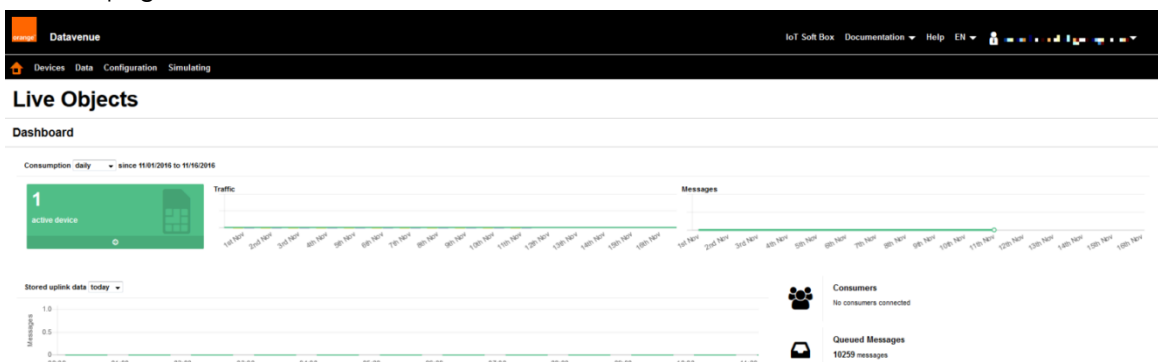
### 3.2.2. Log in

To log in to Live Objects web portal, connect to [liveobjects.orange-business.com](https://liveobjects.orange-business.com) using your web-browser and go into “Sign in”:



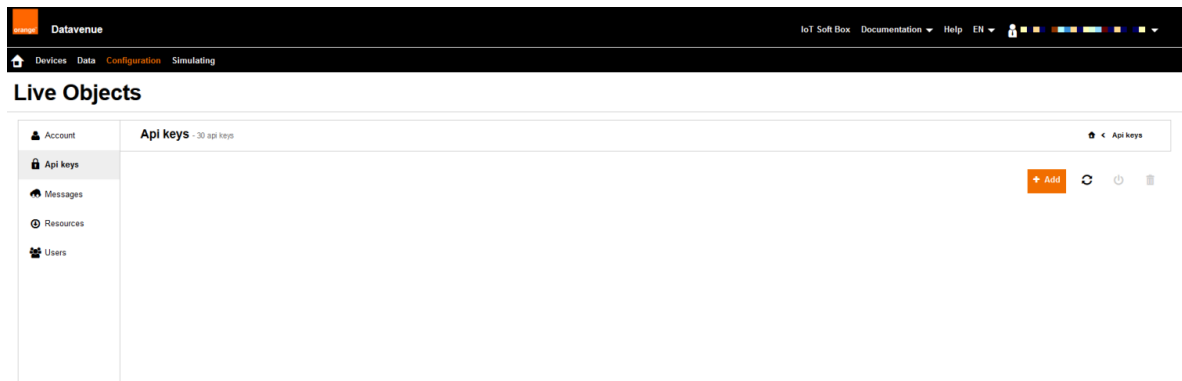
1. Fill the “Log in” form with your credentials:
  - your email address,
  - the password set during the activation phase,
2. Then click on the “Sign in” button.

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 shows the 'Add an api key' form. At the top, there's a header 'Add an api key' with a breadcrumb 'Api keys < Add an api key' and buttons for '+ Add' and 'Back'. The form is divided into two main sections: 'Information' and 'Validity'. The 'Information' section has fields for 'Name' (with a red asterisk indicating it's required), 'Description' (with a red asterisk), and 'Roles' (with a red asterisk). The 'Validity' section has fields for 'From' (with a date '11/17/2016 10:42 AM'), 'To' (with a placeholder 'enter an end date'), and 'TTL' (with a placeholder '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:

1. Connects to [Datavenue Live Objects Platform](#), using:
  - an optional secure connection (TLS)
  - the LiveObjects mode: [Json+Device](#)
2. Publishes
  - The [current Status/Info](#)
  - The [current Configuration Parameters](#)
  - The [current Resources](#)
3. Subscribes to Live Objects topics to receive notifications
  - Configuration Parameters update request
  - Resource update request
  - Command request
4. 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 step 2

### 3.3.2. Packages dependences

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

#### 1) *Mbed-TLS*

<https://github.com/ARMmbed/mbedtls>

Latest commit : cb587009d679812dc27979c867cdc0e056a132e6

#### 2) *Paho MQTT embedded-c*

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

Latest commit : f834fae3d0bb4515c536851536cbe6b5f6f3d1ef

#### 3) *jsmn*

<https://github.com/zserge/jsmn>

Latest commit : 1682c32e9ae5990ddd0f0e907270a0f6dde5cbe9

#### 4) *iotsoftbox-mqtt-core (the latest release of library on github)*

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

Latest Commit : c50131ef0fd84eb6a1d1522d080d639512852282

### 3.3.3. Configure workstation for cross compilation

#### 3.3.3.1. Linux

This example is given for a Linux Debian as a host machine and a Raspberry pi 3 as a target. Instructions for a Raspberry pi 1 are also given.

1. Install the cross compiler

```
$ echo "deb http://emdebian.org/tools/debian/ jessie main" | sudo tee
/etc/apt/sources.list.d/crosstools.list
$ sudo apt install curl
$ curl http://emdebian.org/tools/debian/emdebian-toolchain-archive.key |
sudo apt-key add -
# Raspberry Pi 1
# sudo dpkg --add-architecture armel
$ sudo dpkg --add-architecture armhf
$ sudo apt-get update
# Raspberry Pi 1
# sudo apt-get install git git-flow cmake crossbuild-essential-armel
$ sudo apt-get install crossbuild-essential-armhf
```

#### 3.3.3.2. Windows

This example is given for a Windows 7 as a host machine and a Raspberry pi as a target

1. Install the [compiler](#).
2. Update the cross compiler environment, by downloading the Sysroot. To do so, use  
C:\SysGCC\Raspberry\TOOLS\UpdateSysroot.bat
3. Install [CMake](#).
4. Install [Perl](#) to run CMake properly.

You **MUST** redo step 2 each time you add a library to the target platform.

### 3.3.4. Build

#### 3.3.4.1. Submodule update

You will need to download the third-party libraries by cloning the main library:

<https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-linux>

Two options for that:

1. Using --recursive when cloning the main library.

```
git clone --recursive https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-linux
```

## 2. Using git submodule in the repository

```
git clone https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-linux
cd LiveObjects-iotSoftbox-mqtt-linux
git submodule init
git submodule update
```

After cloning the required library, a repository containing all the packages as well as the examples is created. The basic example liveobjects-sample-basic, which is located in LiveObjects-iotSoftbox-mqtt-linux/examples is the one that will be tested on the device. Then, as you can see later in part 4.2, a modification should be done in the header file **liveobjects\_dev\_params.h** in order to define your API key. Apply the modification then start the building step.

### 3.3.4.2. Linux

#### 1. Local build

```
cd LiveObjects-iotSoftbox-mqtt-linux/
mkdir build
cd build
cmake ..
make
```

#### 2. Cross-Compilation build

```
cd LiveObjects-iotSoftbox-mqtt-linux/
mkdir build
cd build
cmake -DCMAKE_C_COMPILER=<Path to the compiler> ..
make
```

For the raspberry pi 3, the compiler will be “arm-linux-gnueabi-hf-gcc” and for the PI 1 “arm-linux-gnueabi-gcc”

Your binary file will be then created in LiveObjects-iotSoftbox-mqtt-linux/build/bin

### 3.3.4.3. Windows

You can only cross compile the program on Windows, using the following:

```
cd LiveObjects-iotSoftbox-mqtt-linux\script
cmakeWinSetup.bat
make
```

At the end of the building process, your binary file is created in LiveObjects-iotSoftbox-mqtt-linux\script\bin



### 3.3.5. Launch

#### 3.3.5.1. Load the sample on your Raspberry Pi

- 1) Make sure your Raspberry Pi is connected to the internet
- 2) Use a Secure CoPy (scp) tool to copy the executable (if you're using [Windows](#)) on the board.

```
cd LiveObjects-iotSoftbox-mqtt-linux/build/bin  
scp -r -p basic <username>@<IP_Address>:destination_path
```

- 3) If you're using linux, an ssh connexion should be established. So first, you should have the username, the password and the IP address of your device. Then, in a new terminal, type the following command then type the password when asked:

```
ssh <username>@<IP_Address>
```

#### 3.3.5.2. Execute

In case you are working on a Windows platform you have to consider the permissions issues. So, on your Raspberry Pi, go to the program's directory and add the permission to execute it:

```
sudo chmod a+x <your binary name>
```

Finally you can run the executable:

```
./<your binary name>
```

in the correct directory of the device.

#### 3.3.5.3. Application Monitoring/Testing

There are 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, send commands, update resources and more.





## 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 Linux boards is available [here](#), library called **LiveObjects-iotSoftbox-mqtt-linux**.

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:

- IP Address: 84.39.42.214
- 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 user header file **liveobjects\_dev\_params.h**:

- Tenant *ApiKey* parameter
- if the TLS is enabled (value : 1) or not (value : 0)

```

/* Set to 1 to enable TLS feature*/
/* (warning: check that LOC_SERV_PORT is the correct port in this case)*/
#define SECURITY_ENABLED                1

/** 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

```

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

However 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`:

```

/* When set to 1, use the MAC address as LiveObject device identifier*/
/* otherwise use LOC_CLIENT_DEV_ID*/
#define LOC_CLIENT_USE_MAC_ADDR          0

/* Default LiveObjects device settings : name space and device identifier*/
#define LOC_CLIENT_DEV_NAME_SPACE        "LiveObjectsDomain"
#if !LOC_CLIENT_USE_MAC_ADDR
#define LOC_CLIENT_DEV_ID                "LO_softboxlinux_01"
#endif

```

If `LOC_CLIENT_USE_MAC_ADDR` is set to 1, the iotsoftbox-mqtt library uses the physical network address (Ethernet MAC address, ..) for the device identifier, otherwise the device identifier is defined by `LOC_CLIENT_DEV_ID`

## 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 sample running with Linux uses the multi-thread model.

## 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 "LINUX BASIC SAMPLE V01.01"
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:


 Datavenue


IoT Soft Box Documentation Help EN 


 Devices Data Configuration Simulating

## Live Objects


 Status

 Parameters


 Commands


 Resources


LiveObjectsSample / LO\_sample\_dev01

 < Managed < LiveObjectsSample / LO\_sample\_dev01


Device Id	LO_sample_dev01
Device Namespace	LiveObjectsSample
Status	<div><div></div>connected</div>
mqttTimeout	30
mqttConnStart	2016-11-02T13:15:24.864Z
mqttVersion	4
apiKeyId	580ddb160d22e0747c14b9e
mqttUsername	json+device
sample_version	MBED SAMPLE V01.05
sample_counter	0
sample_message	READY
Last contact	a few seconds ago
Topic for parameters updates	pubsub/~f4de43fd0d842049a45019d064d839b
Topic for commands updates	pubsub/~21685167935645ad993c0b152ee3e5c5
Topic for resources updates	pubsub/~7a966613438cd48cc9471012dee0d3445


 **Datavenue**


IoT Soft Box Documentation ▾ Help EN ▾ 


 **Devices** Data Configuration Simulating

## Live Objects


 Status

 Parameters

 Commands

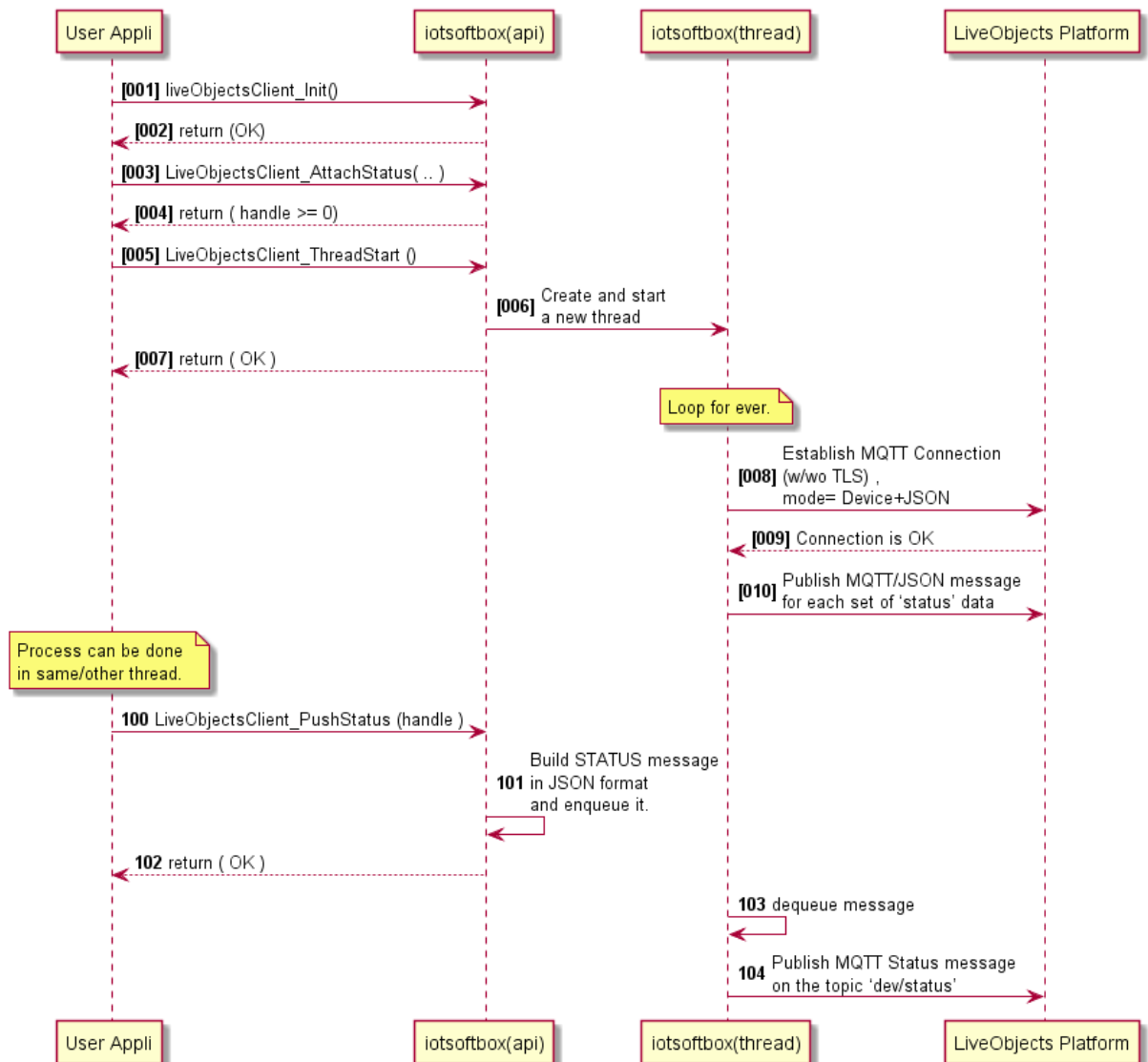
 Resources

LiveObjectsSample / LO\_sample\_dev01

 < Managed < LiveObjectsSample / LO\_sample\_dev01

Device Id	LO_sample_dev01																
Device Namespace	LiveObjectsSample																
Status	<div><div><div></div>connected</div></div> <table><tr><td>mqttTimeout</td><td>30</td></tr><tr><td>mqttConnStart</td><td>2016-11-02T13:15:24.864Z</td></tr><tr><td>mqttVersion</td><td>4</td></tr><tr><td>apiKeyId</td><td>580ddb160cf22e0747c14b9e</td></tr><tr><td>mqttUsername</td><td>json+device</td></tr><tr><td>sample_version</td><td>MBED SAMPLE V01.05</td></tr><tr><td>sample_counter</td><td>0</td></tr><tr><td>sample_message</td><td>READY</td></tr></table>	mqttTimeout	30	mqttConnStart	2016-11-02T13:15:24.864Z	mqttVersion	4	apiKeyId	580ddb160cf22e0747c14b9e	mqttUsername	json+device	sample_version	MBED SAMPLE V01.05	sample_counter	0	sample_message	READY
mqttTimeout	30																
mqttConnStart	2016-11-02T13:15:24.864Z																
mqttVersion	4																
apiKeyId	580ddb160cf22e0747c14b9e																
mqttUsername	json+device																
sample_version	MBED SAMPLE V01.05																
sample_counter	0																
sample_message	READY																
Last contact	a few seconds ago																
Topic for parameters updates	pubsub/~fade43fbddd842849a45019df64d839b																
Topic for commands updates	pubsub/~21685167935645ad993cfb152ee3e5c5																
Topic for resources updates	pubsub/~7a966613438d48cc9471012dea0d3445																

#### 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 identifier 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->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.

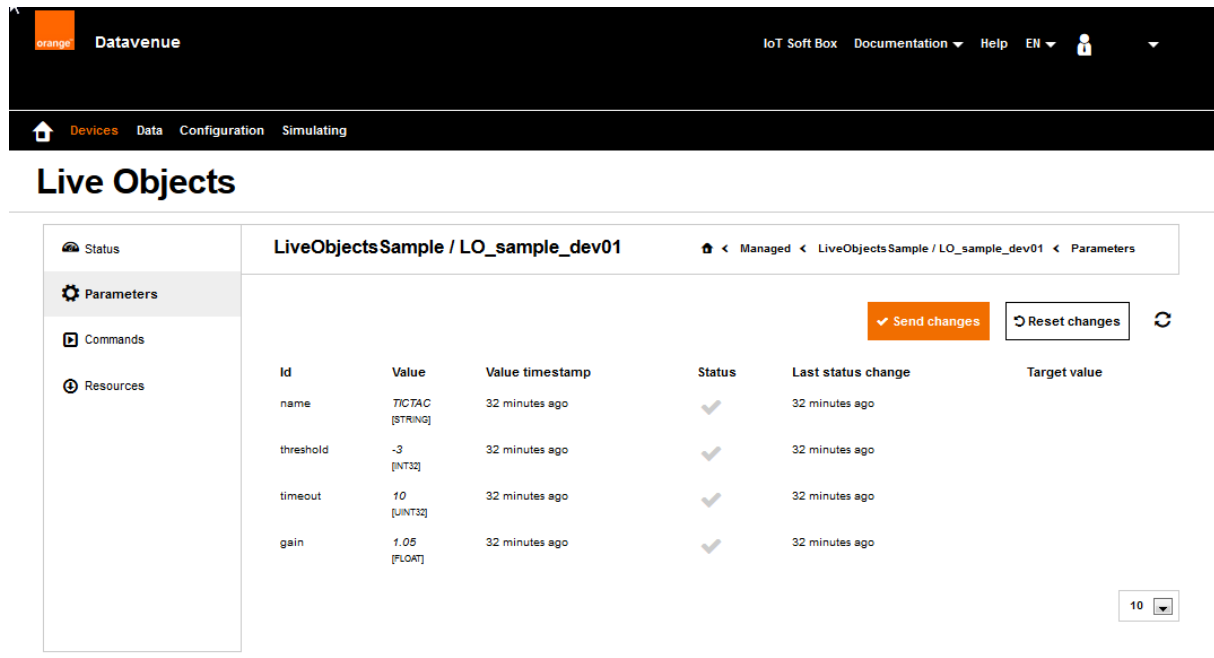
### 4.6.2. Push a set of configuration parameters

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:



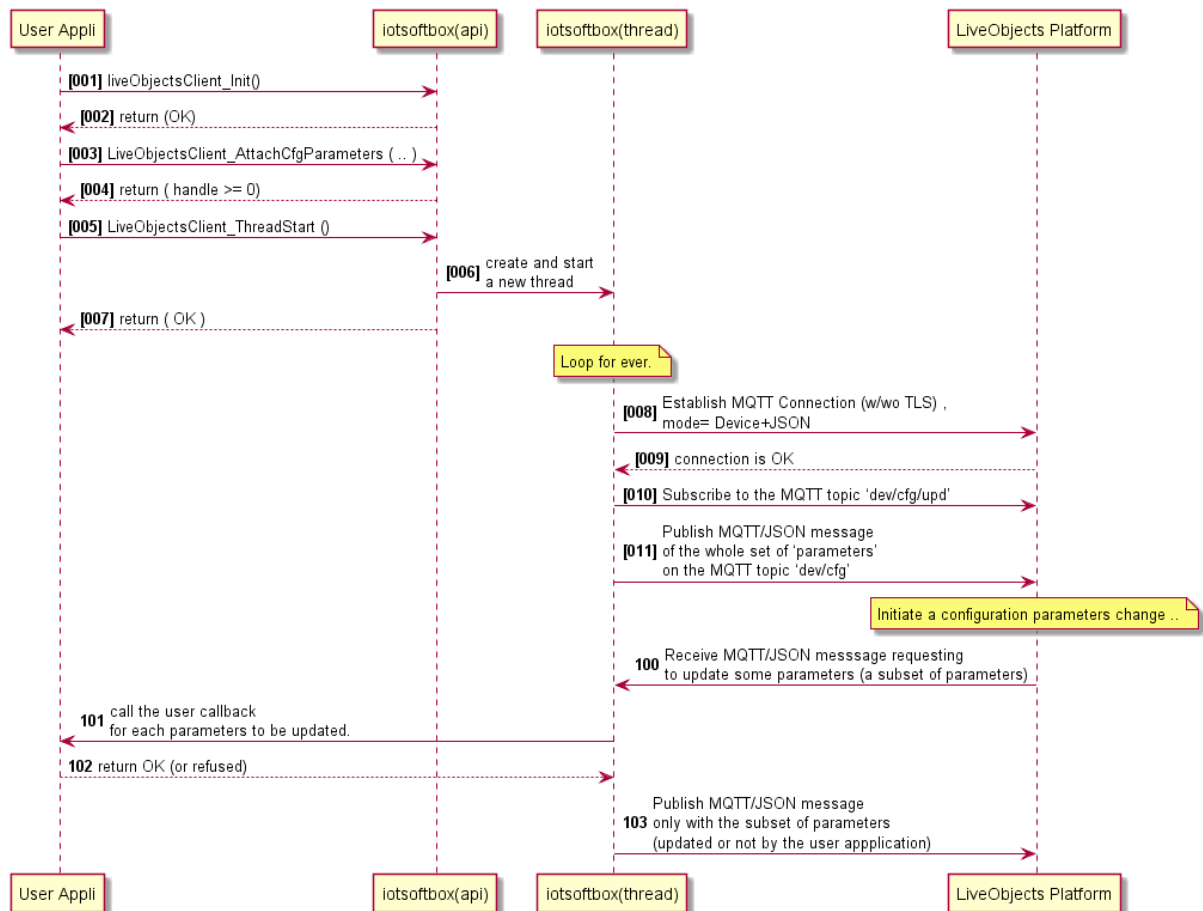
The screenshot shows the Datavenue Live Objects portal interface. The top navigation bar includes the Datavenue logo, user information, and links to IoT Soft Box, Documentation, Help, and EN. The main navigation bar has tabs for Devices, Data, Configuration, and Simulating. The 'Live Objects' section is active, displaying a list of parameters for the device 'LiveObjectsSample / LO\_sample\_dev01'.

On the left, a sidebar contains links for Status, Parameters (selected), Commands, and Resources. The main content area shows a table of parameters with columns: Id, Value, Value timestamp, Status, Last status change, and Target value. The parameters listed are name, threshold, timeout, and gain, all with a status of 'OK' and a last status change of '32 minutes ago'.

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

At the top right of the parameters table, there are buttons for 'Send changes' (orange) and 'Reset changes' (white), along with a refresh icon. A dropdown menu at the bottom right shows the number '10'.

#### 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_tem, 1p },
    { LOD_TYPE_FLOAT, "battery_level", &appv_measures_volt, 1 }
};
#define SET_MEASURES_NB (sizeof(appv_set_measures) / sizeof(LiveObjectsD_Data_t))
```

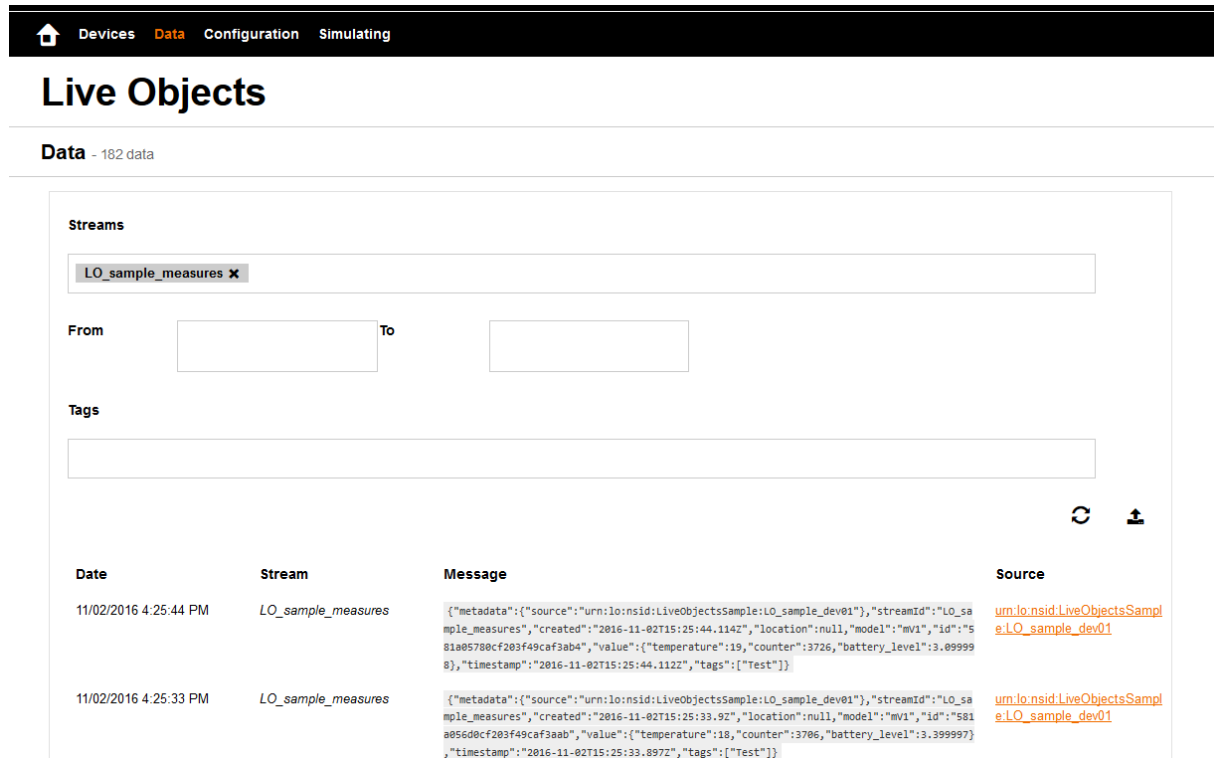
#### 4.7.2. Push the set of collected data

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

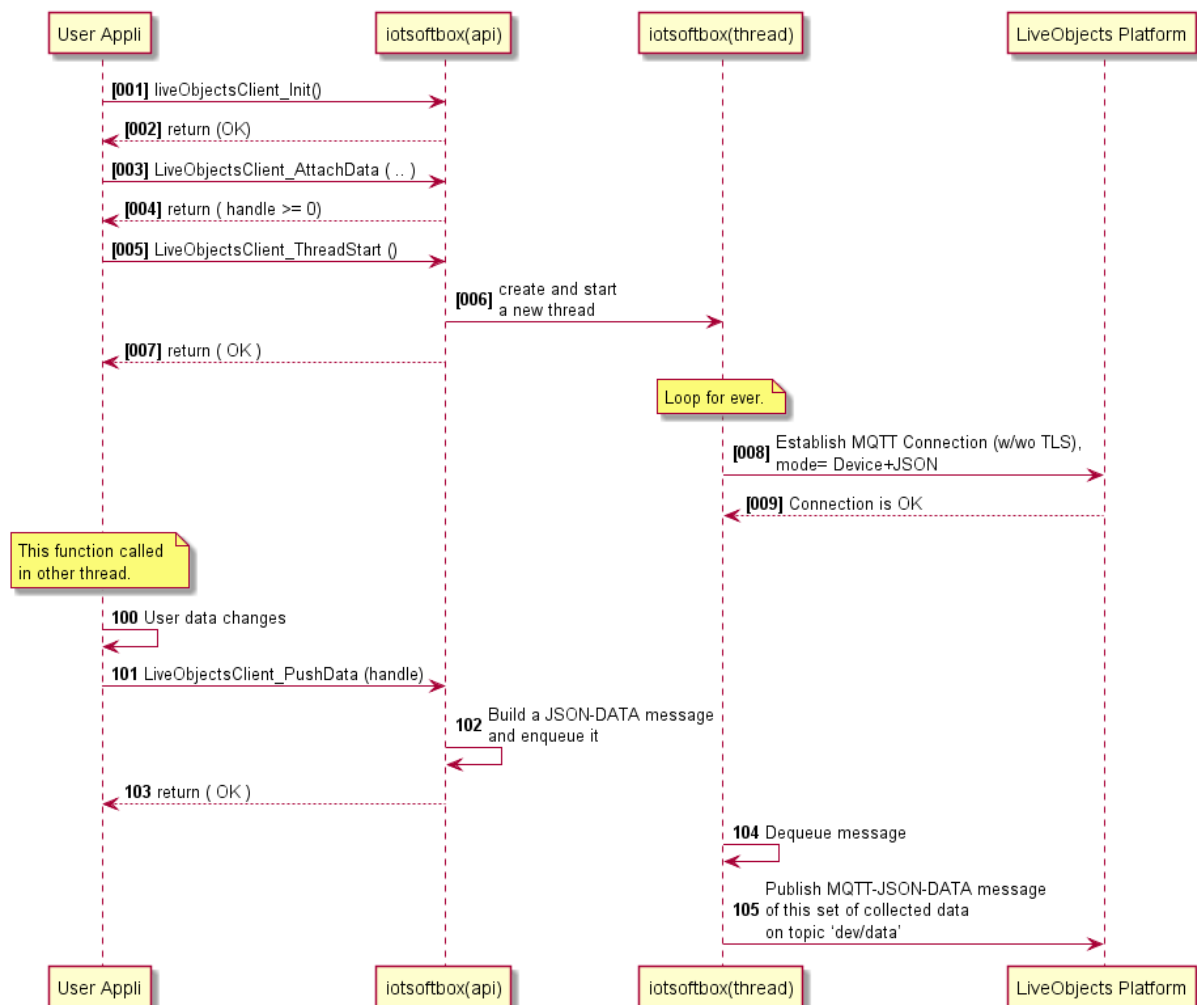
On the Datavenu Live Objects portal, the user can check the 'Collected Data' published by its connected device (here, filter is set to get only stream = LO\_sample\_measures):



The screenshot shows the 'Live Objects' portal interface. At the top, there is a navigation bar with 'Devices', 'Data', 'Configuration', and 'Simulating'. Below this, the 'Live Objects' title is displayed. A 'Data' section indicates '182 data'. The 'Streams' section shows a filter 'LO\_sample\_measures' with a close button. Below the filter, there are 'From' and 'To' input fields. The 'Tags' section has an empty input field. On the right, there are icons for refresh and download. The main data table has four columns: Date, Stream, Message, and Source. It contains two rows of data, both for the 'LO\_sample\_measures' stream, with timestamps from 11/02/2016 4:25:44 PM and 11/02/2016 4:25:33 PM. The 'Message' column contains JSON metadata and sensor data like temperature, counter, and battery level. The 'Source' column provides a URL to the specific data point.

Date	Stream	Message	Source
11/02/2016 4:25:44 PM	LO_sample_measures	<pre>{   "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": "581a95798cf203f49caf3ab4",     "value": {       "temperature": 19,       "counter": 3726,       "battery_level": 3.099998     },     "timestamp": "2016-11-02T15:25:44.112Z",     "tags": ["Test"]   } }</pre>	<a href="urn:lo:nsid:LiveObjectsSample:LO_sample_dev01">urn:lo:nsid:LiveObjectsSample:LO_sample_dev01</a>
11/02/2016 4:25:33 PM	LO_sample_measures	<pre>{   "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": "581a956d0cf203f49caf3a3b",     "value": {       "temperature": 18,       "counter": 3786,       "battery_level": 3.399997     },     "timestamp": "2016-11-02T15:25:33.897Z",     "tags": ["Test"]   } }</pre>	<a href="urn:lo:nsid:LiveObjectsSample:LO_sample_dev01">urn:lo:nsid:LiveObjectsSample:LO_sample_dev01</a>

#### 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

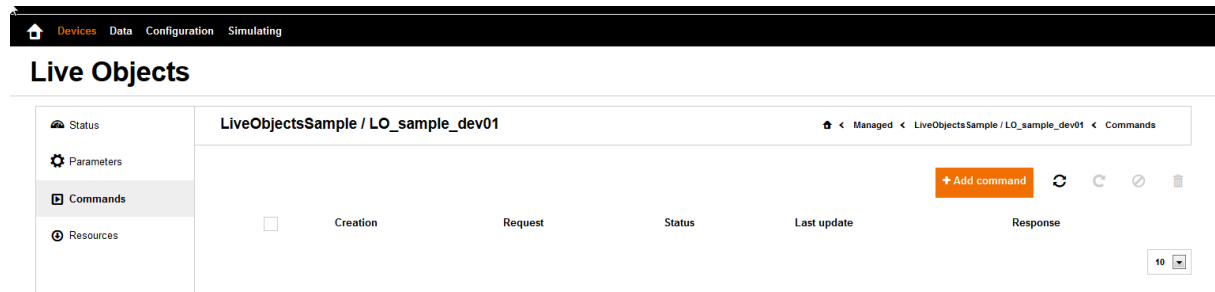
As soon as the device is ready (or not) to process commands, the application can enable (or disable) the 'command' feature by calling the function:

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

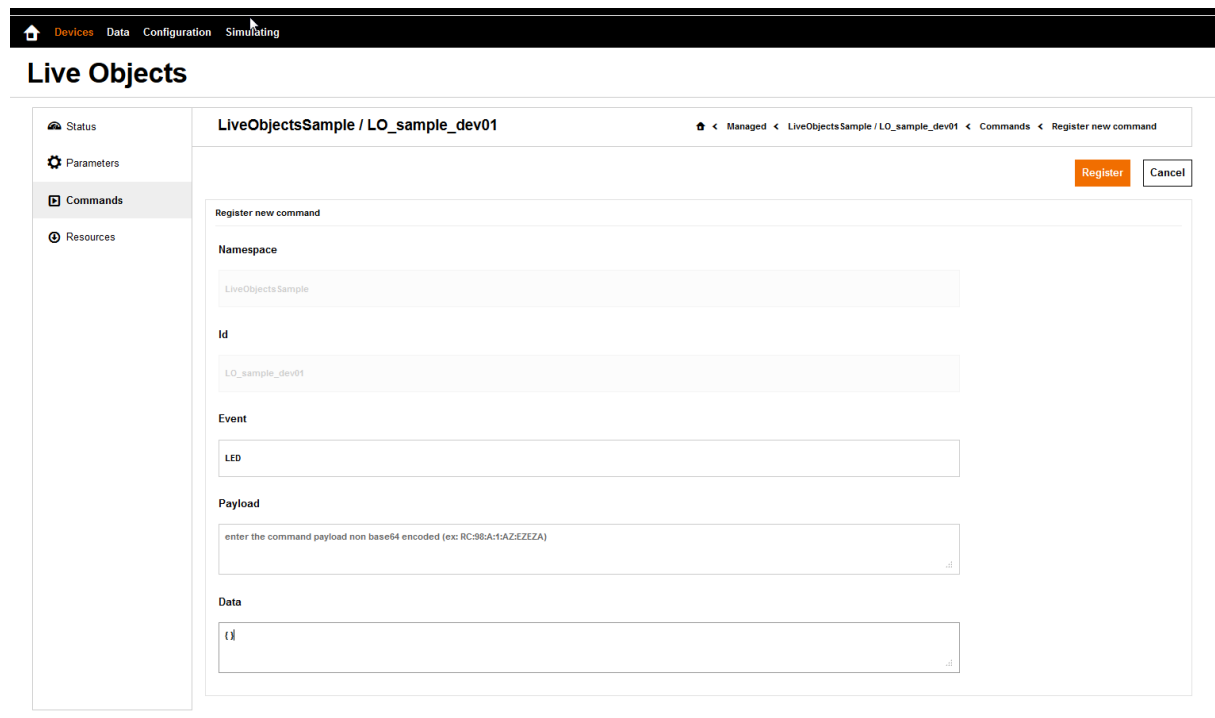
### 4.8.3. Use of Live Objects Portal to send a command

On the Live Objects Portal,

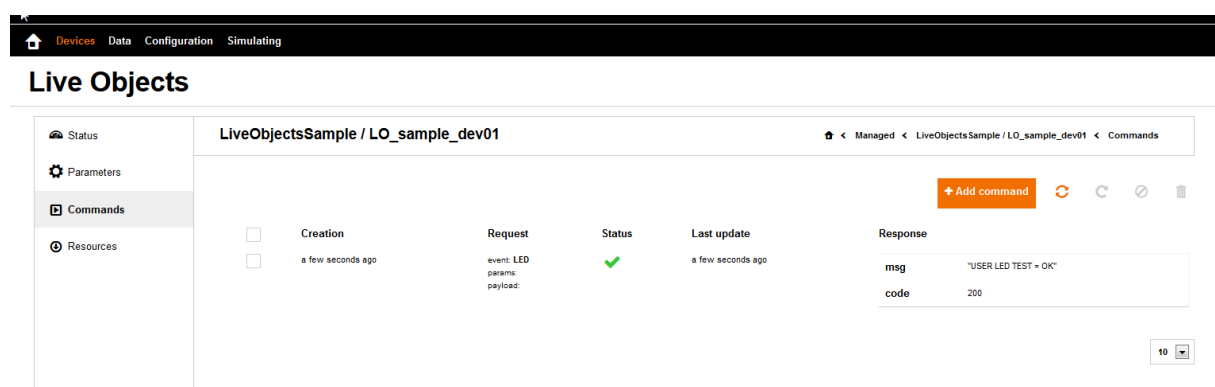
- Go to tab: Devices -> <your device> -> Commands



- Click on button '+ Add command'

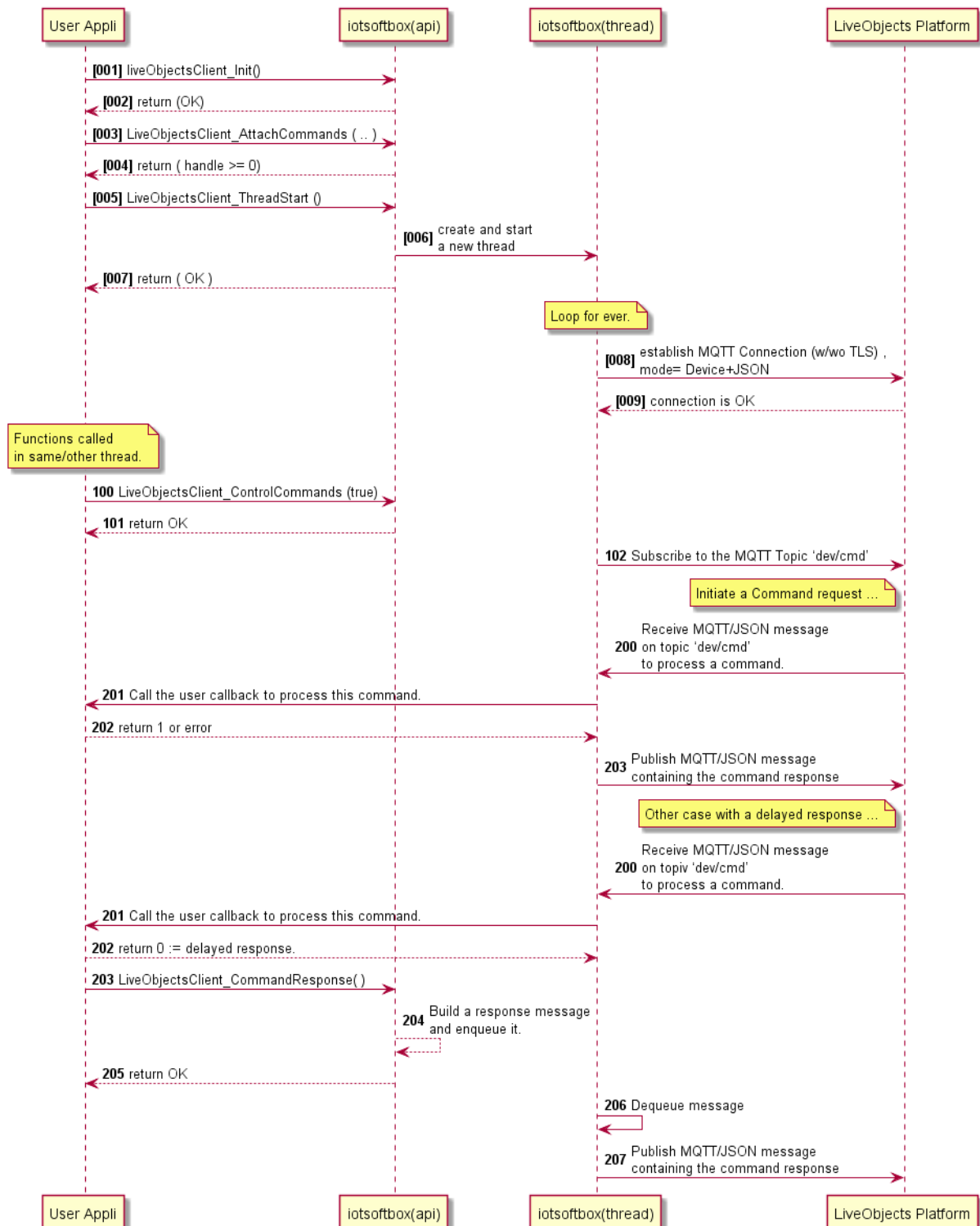


- Click on button 'Register'. And wait a few moment , refresh the web page





#### 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 ntfcB,
    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. `resourceCb`) 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 several time if you're using a very low capacity device. Once you have the data you can process them.

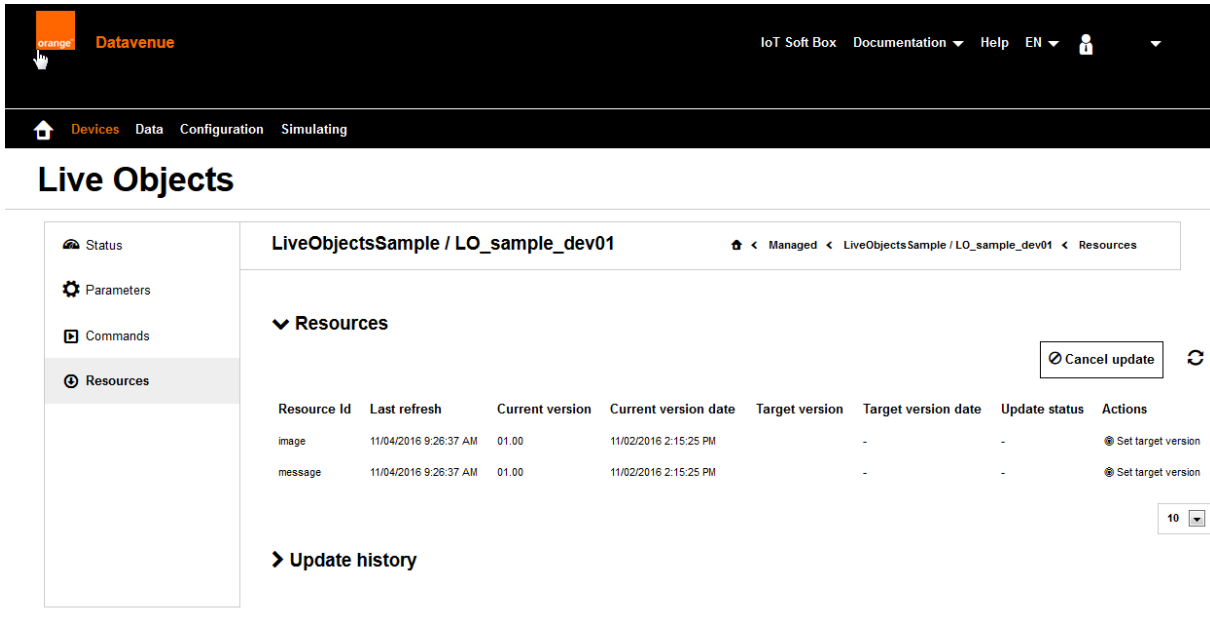
#### 4.9.2. Enable/disable 'resources' feature

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.



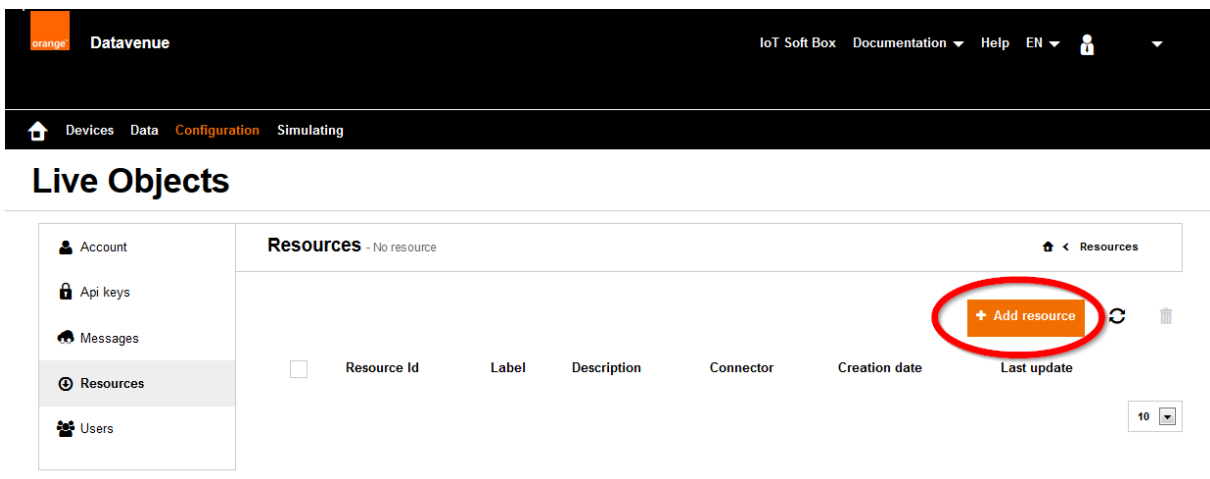
The screenshot shows the 'Live Objects' portal interface. The top navigation bar includes 'Datavenue' and links for 'IoT Soft Box', 'Documentation', 'Help', 'EN', and a user profile. The main navigation bar has 'Devices', 'Data', 'Configuration', and 'Simulating'. The 'Live Objects' section is active, showing a sidebar with 'Status', 'Parameters', 'Commands', and 'Resources'. The main content area displays the 'Resources' tab for the device 'LiveObjectsSample / LO\_sample\_dev01'. A table lists the 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

Buttons for 'Cancel update' and a refresh icon are visible. An 'Update history' link is at the bottom.

Here, the device 'LiveObjectsSample/LO\_sample\_dev01' has two resources identified by: *image* and *message*.

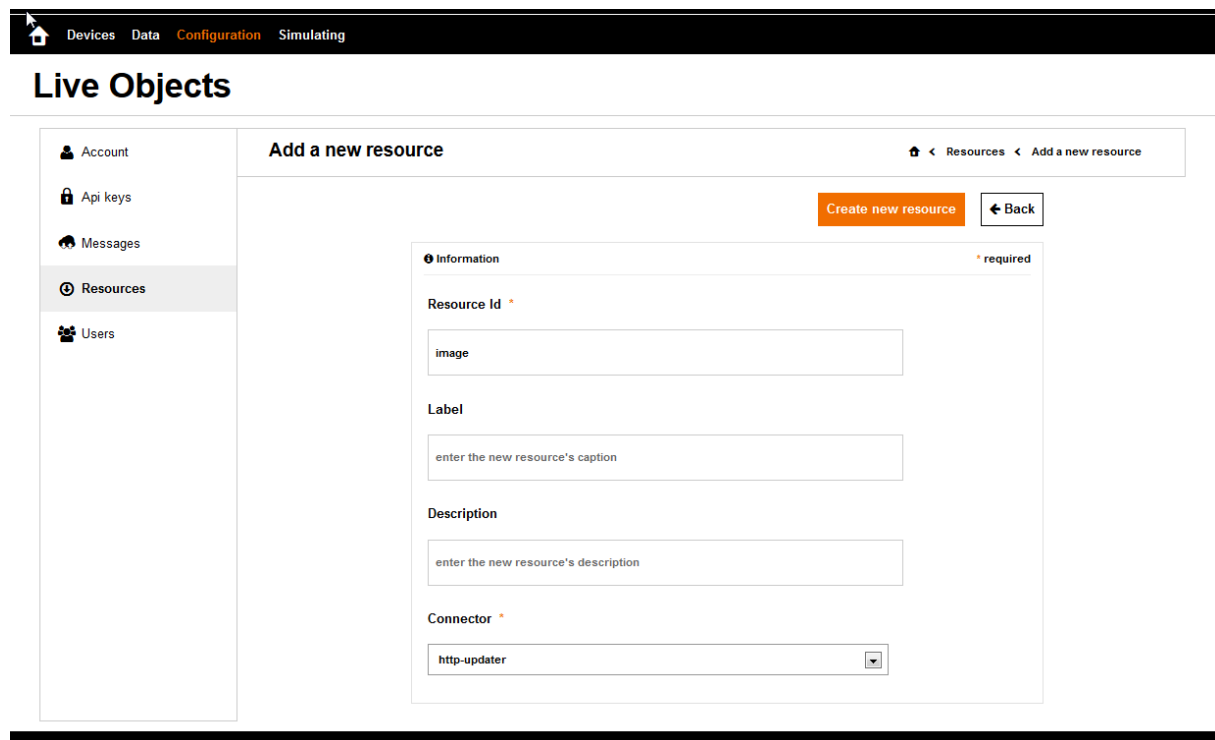
Now, the user can create a new resource on the Live Objects platform, in the tab 'Configuration->Resources', associated to these resources



The screenshot shows the 'Live Objects' portal interface. The top navigation bar includes 'Datavenue' and links for 'IoT Soft Box', 'Documentation', 'Help', 'EN', and a user profile. The main navigation bar has 'Devices', 'Data', 'Configuration', and 'Simulating'. The 'Live Objects' section is active, showing a sidebar with 'Account', 'Api keys', 'Messages', 'Resources', and 'Users'. The main content area displays the 'Resources' tab, which is currently empty with the text 'Resources - No resource'. A red circle highlights the '+ Add resource' button in the top right corner of the main content area.

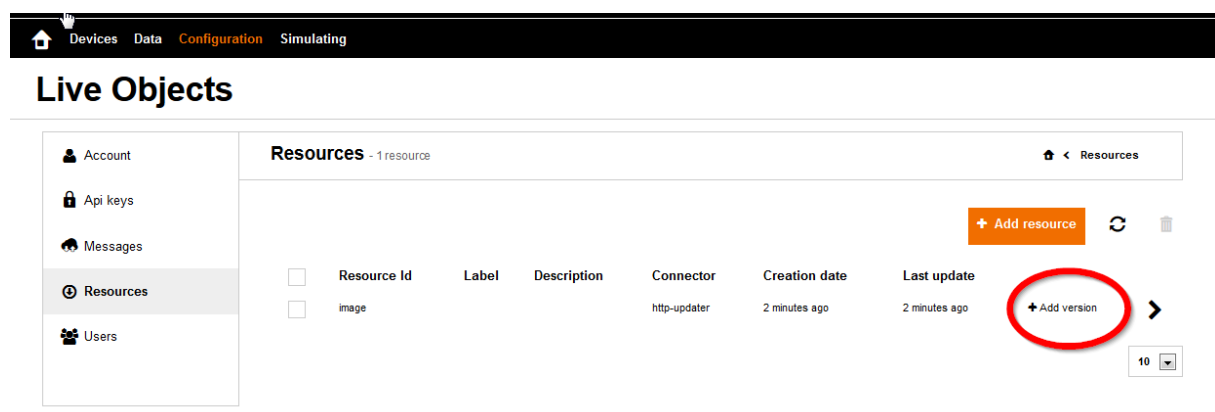
Two fields are mandatory

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



The screenshot shows the 'Live Objects' interface with a sidebar on the left containing 'Account', 'Api keys', 'Messages', 'Resources' (selected), and 'Users'. The main area is titled 'Add a new resource' with a breadcrumb 'Resources < Add a new resource'. There are two buttons: 'Create new resource' (orange) and 'Back' (white with a left arrow). Below these is a form titled 'Information' with a '\* required' label. The form contains four fields: 'Resource Id' (with a sub-label 'image'), 'Label' (with placeholder text 'enter the new resource's caption'), 'Description' (with placeholder text 'enter the new resource's description'), and 'Connector' (a dropdown menu with 'http-updater' selected).

The result is the following:

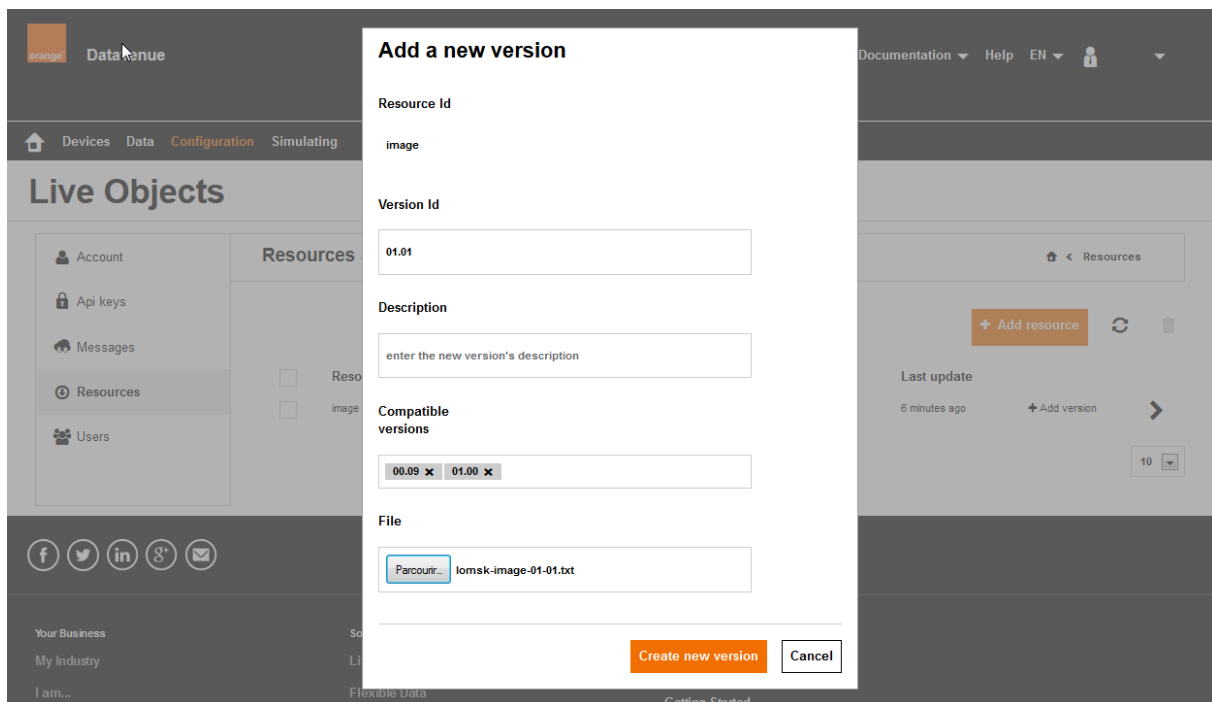


The screenshot shows the 'Live Objects' interface with the 'Resources' section selected. The main area is titled 'Resources - 1 resource' with a breadcrumb 'Resources < Resources'. There are two buttons: 'Add resource' (orange with a plus icon) and a refresh icon. Below these is a table with the following columns: 'Resource Id', 'Label', 'Description', 'Connector', 'Creation date', and 'Last update'. The table contains one row with the following data: 'image', 'http-updater', '2 minutes ago', and '2 minutes ago'. To the right of the table is a red circle around a '+ Add version' button, followed by a right arrow button. At the bottom right is a pagination control showing '10' and a dropdown arrow.

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

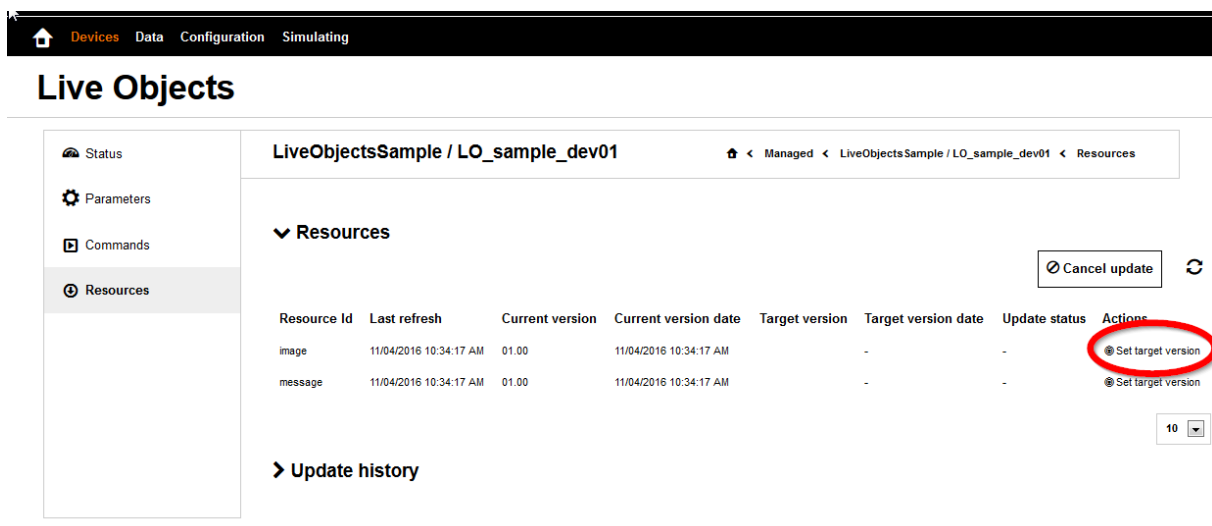
Then, a new resource version can be attached to this resource 'image', by specifying:

- **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

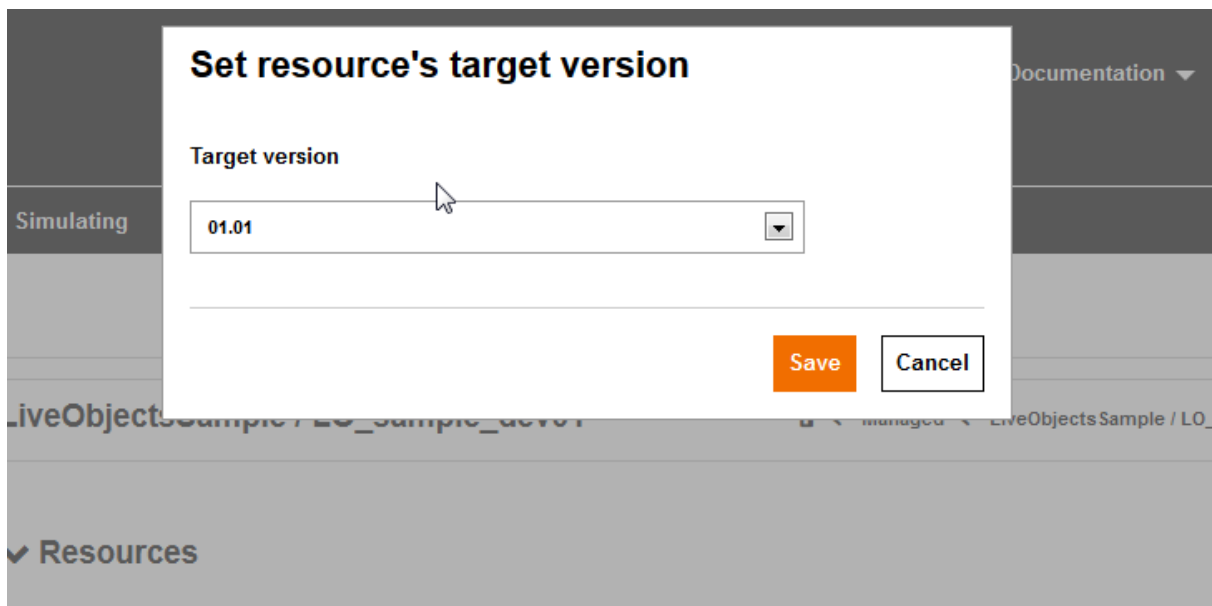


Now, the resource update request can be launched for this device:

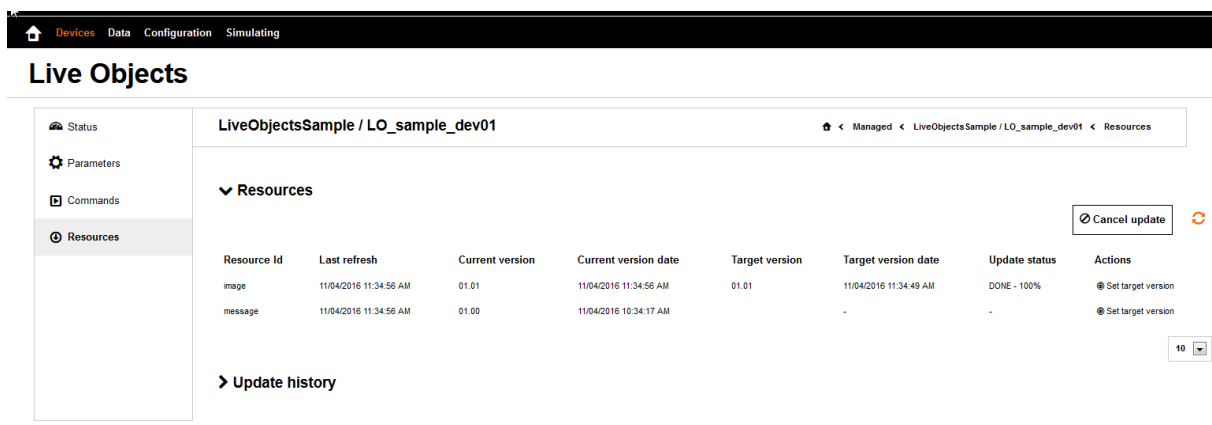
- Go to the 'devices' tab.
- Select your connected device, here it is "LiveObjectsSample / LO\_sample\_dev01"
- Go to the 'resources' tab



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

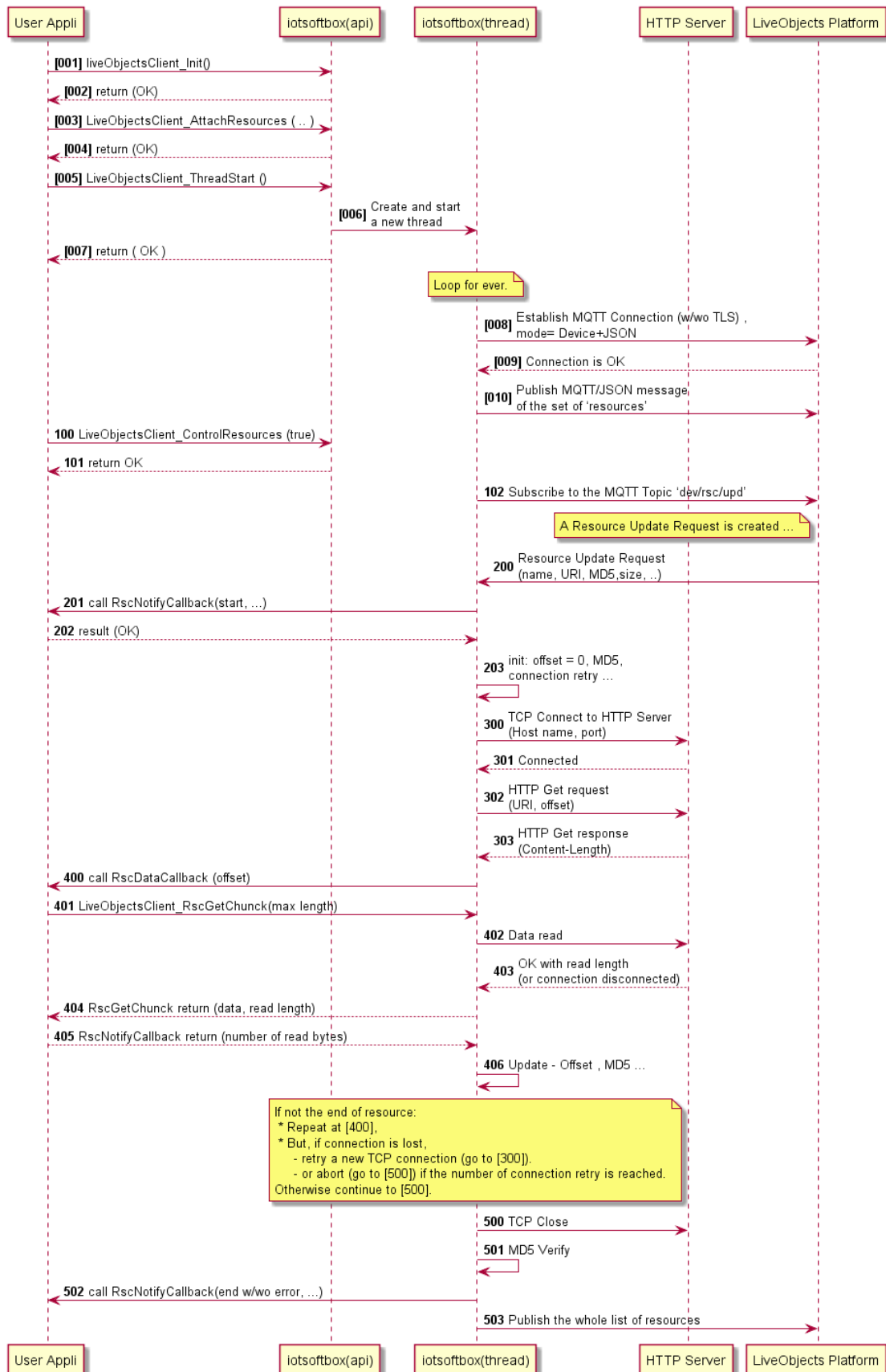


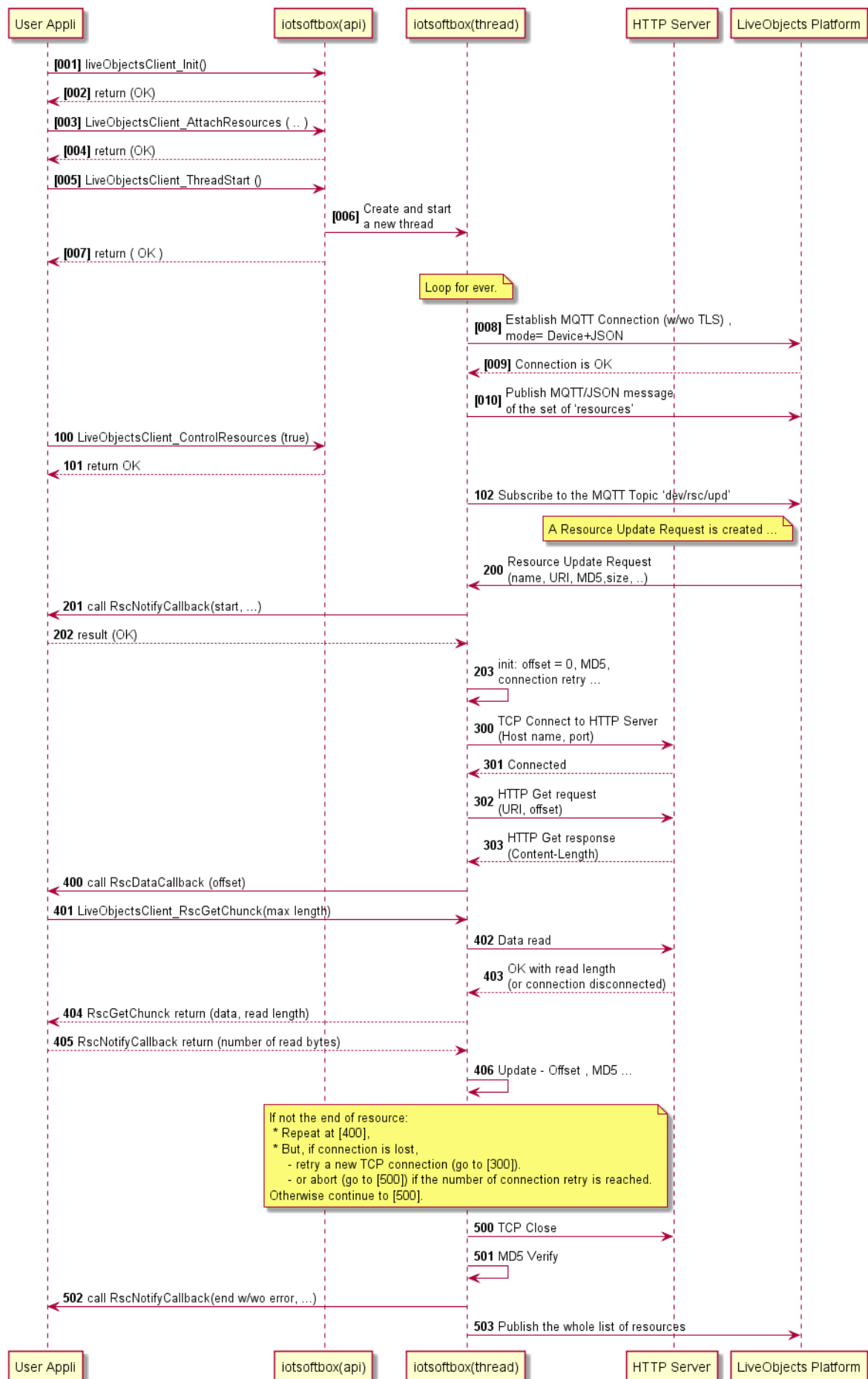
At the end of transfer, after refreshing the web page, the current version should be equal to the target version:





#### 4.9.4. Sequence diagram







An example of URI is:

<http://liveobjects.orange-business.com:80/dl/18p1bj775jkh0pi6p49076hk45>

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
```

## 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 under Linux use the following command:

```
doxygen mqtt_live_objects/iotsoftbox-mqtt-core/doxygen/liveobjects_iotsoftbox.doxyfile
```

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

You can also use a GUI (on linux it's called doxywizard).

Step 1: Specify the working directory.

Step 2: Stay in the **Wizard** tab, fill the fields from the **Project** topic, then go to the output topic, uncheck **LaTeX** unless you want it. then go to the **Run** tab and press **Run Doxygen**.

The doc will be in the directory you specified in the **Project** topic.

### 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\_INF
- LOTRACE\_DBG
- LOTRACE\_DBG\_VERBOSE

#### 5.2.1. Linux environment

The LOTRACE\_XXX MACROs are mapped onto printf function defined in <stdio.h>

As an alternative you can log everything into the syslog of your device. To do so, go into **liveobjects\_dev\_param.h** and switch **#define SYSLOG 0** to **1**.

## 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_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 1`  
Support or not the Live Objects 'resources' feature.
- `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_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_DATA_SET`  
Max Number of collected data streams (or also named 'data sets') (default: 5 data streams)
- `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)