IoT Soft Box Starter Kit

User Manual for iotsoftbox-mqtt library

orange™

Table of contents

1. Introduction

1.1. Document purpose

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

- Overview
- Getting started
- Features
- Usefull links

1.2. Reference documents

#	Origin	Title
1	Orange	Datavenue Live Objects - complete guide (1.4.1.)
2	ARM mbed	FRDM-K64F devlopment board
3	ARM mbed	mbed documentation

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. ARM mbed OS

ARM mbed OS is an open source embedded operating system designed specifically for the "things" in the Internet of Things.

It includes robust security foundations; standard based communication capabilities, drivers for sensors, I/O devices and connectivity. mbed OS is built as a modular, configurable software stack so it can be customized for the device used for development.

To find out more about the mbed OS:

- https://developer.mbed.org/blog/entry/Introducing-mbed-OS-5/
- https://developer.mbed.org/

Live Objects iotsoftbox-mqtt library is fully compatible with the ARM mbed 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.

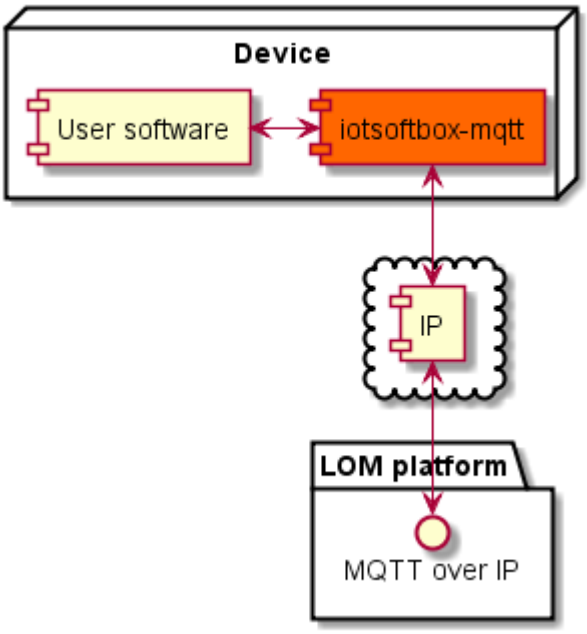


Figure 1 – IoT Soft Box integration in a system

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, 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.
- JSM, a simple C library only used to parse the received JSON messages. The JSMN is available here.
- Mbed TLS (already included in mbed OS 5.0 and later)

3. Getting started

3.1. Hardware Environment

To test our SDK with a compatible hardware, please us:

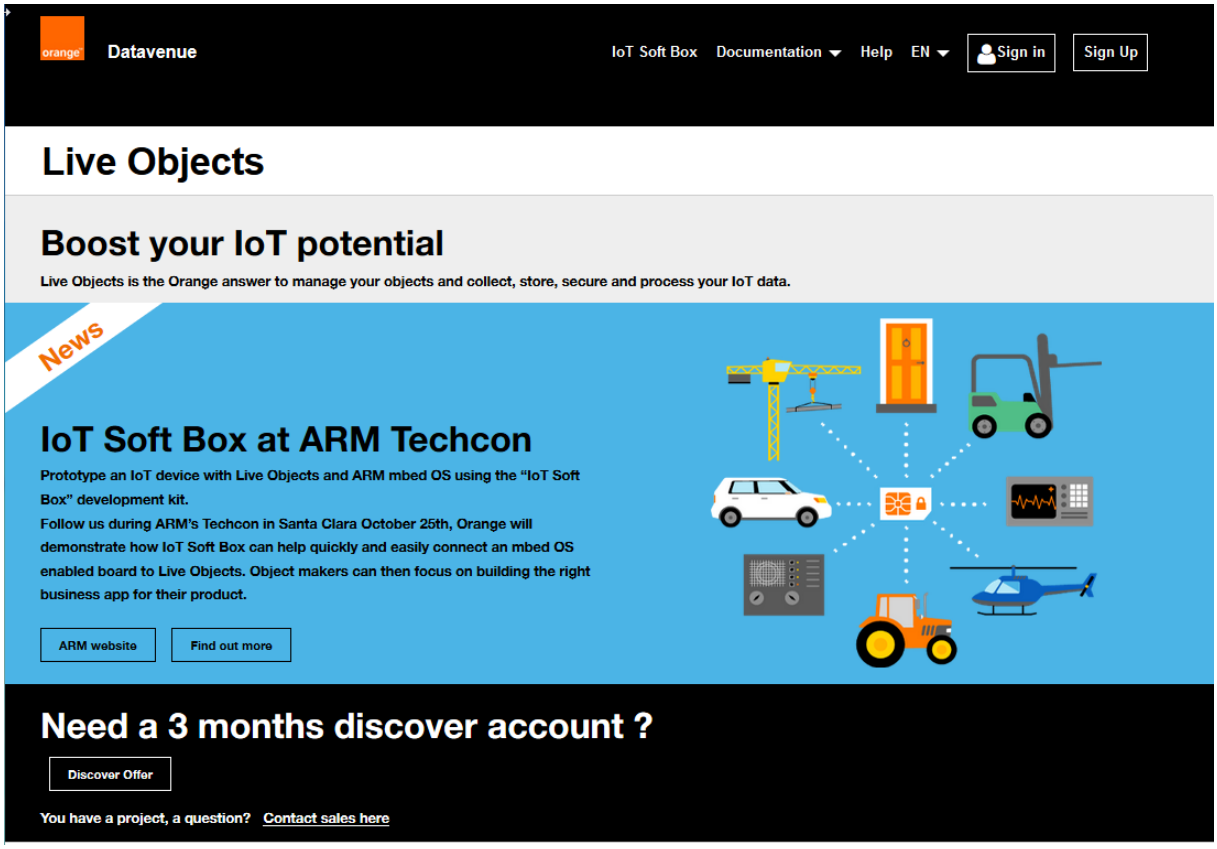
- Our IoT Soft Box hardware kit (available soon for online purchase)
  or
- The NXP development board : Freescale K64F

3.2. Acces to Live Objects

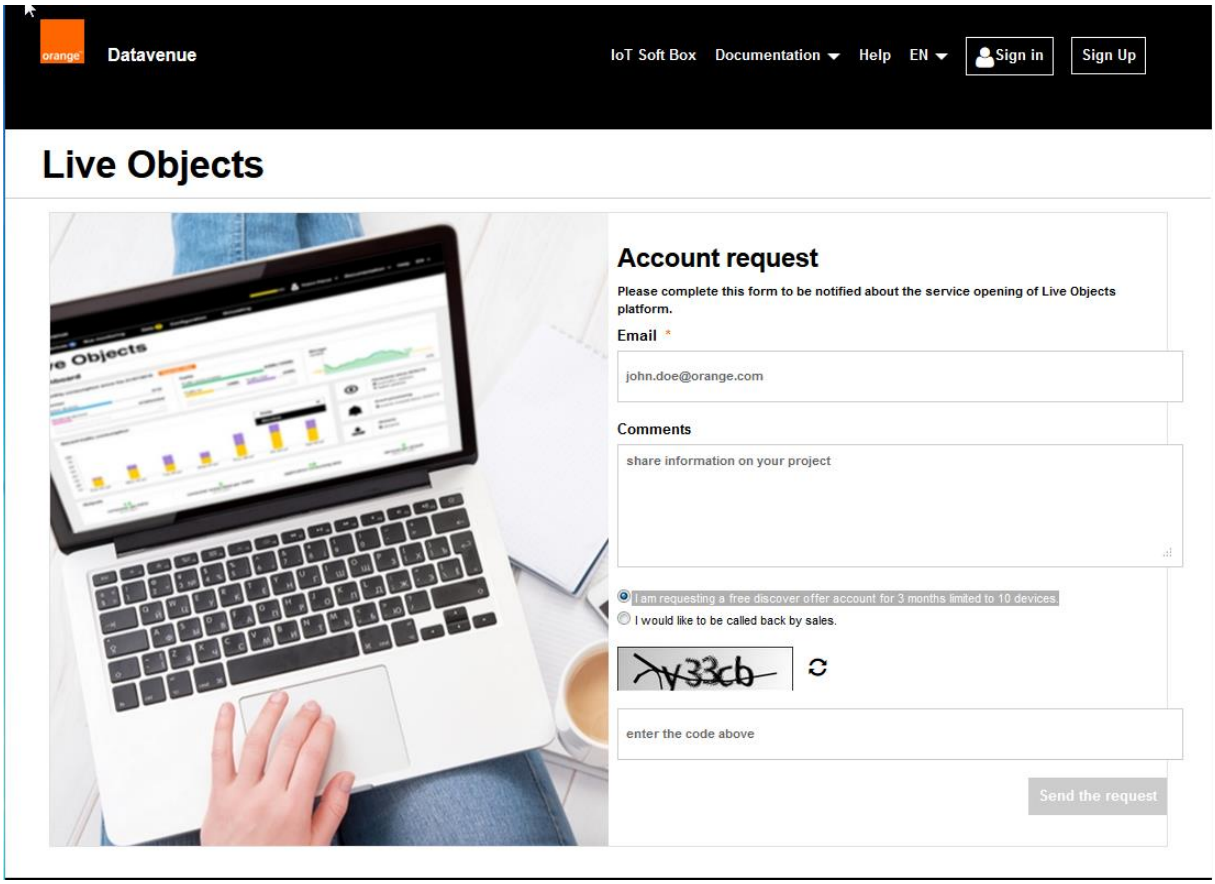
3.2.1. Account creation

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/).



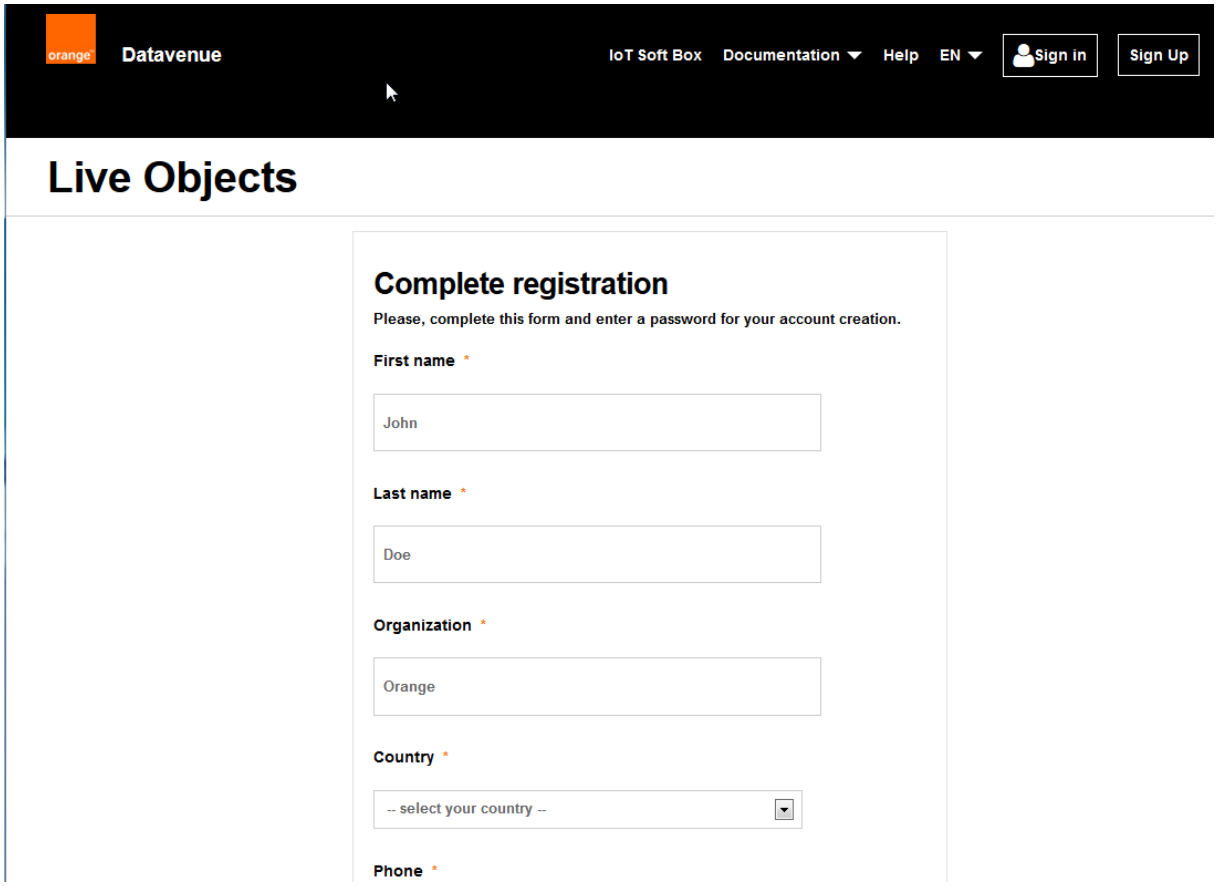
2. Click on '**Discover Offer**' button (or Sign Up) and fill the form, checking option 'I am requesting a free discover offer account for 3 months limited to 10 devices'.



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.

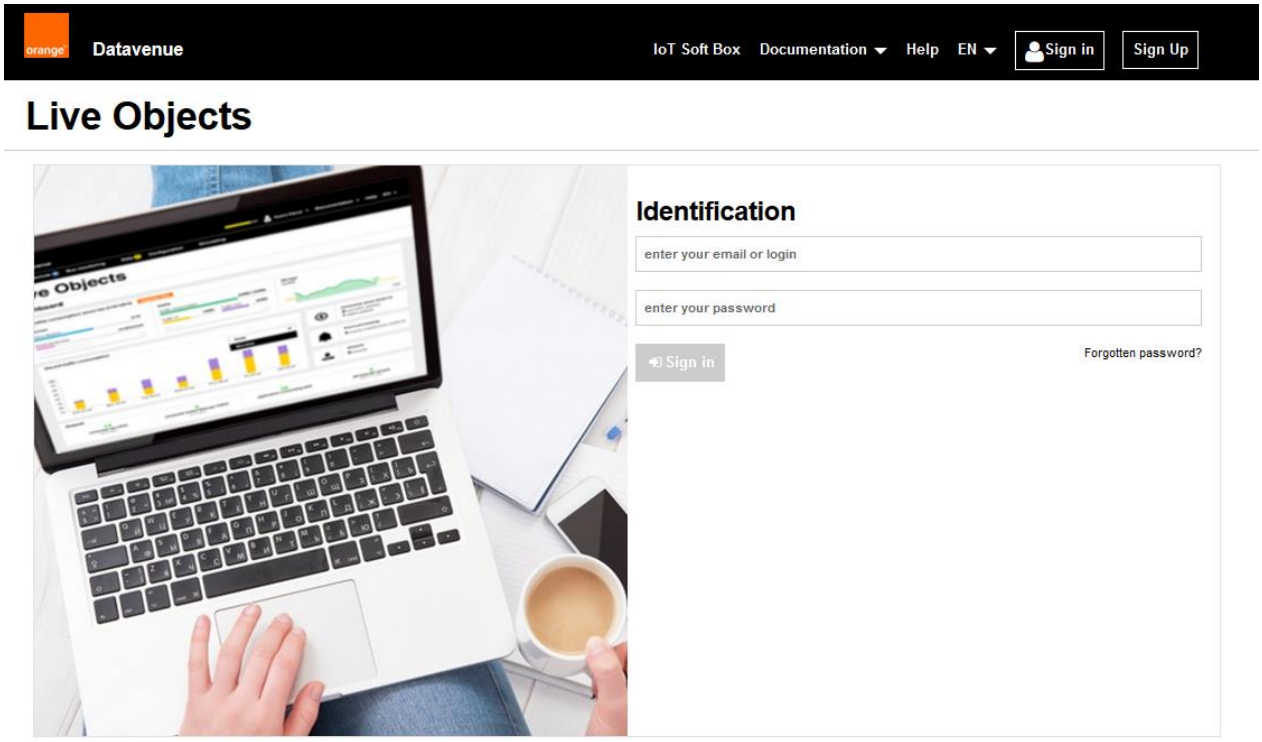
4. Follow the link, fill the form, and click on ‘Validate’.



1. 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:

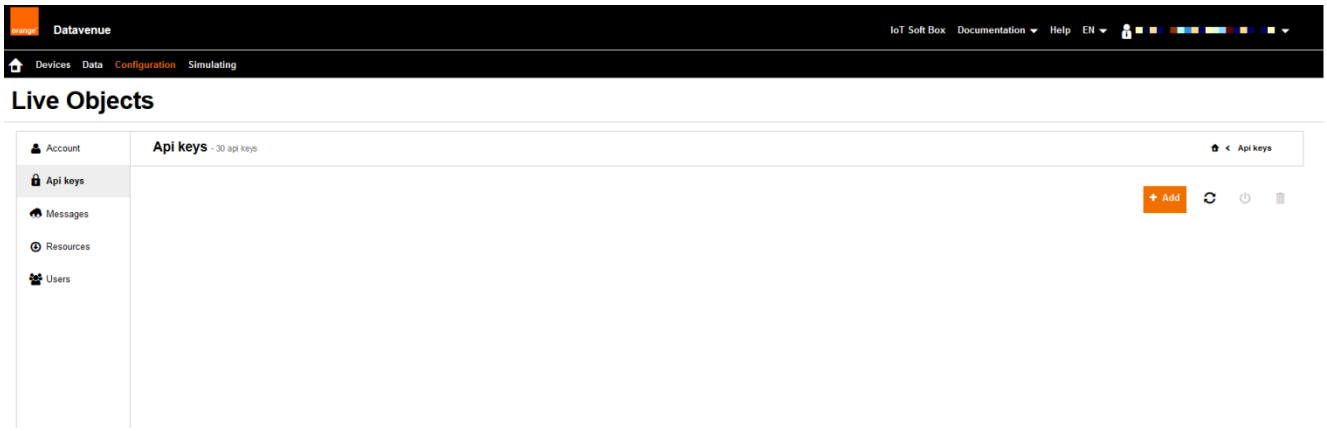


1. Fill the “Log in” form with your credentials:
    - your email address,
    - the password set during the activation phase,
2. Then click on the “Log in” button.

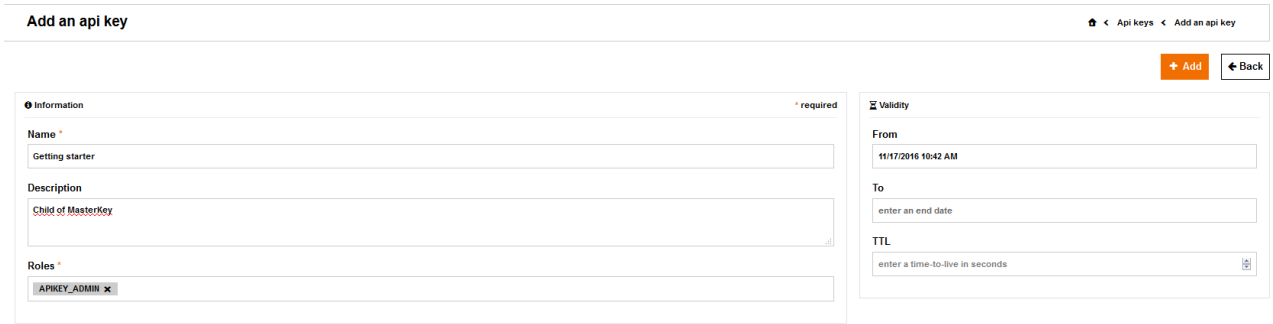
If the credentials are correct, a success message is displayed and you are redirected to your “home” page:

3.2.3. API Key creation

To get a device or an application communicating with Live Objects Manage, 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 Manage.



As a security measure, you can not 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.



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 IoTexamples.

When running on development board, the embedded ‘basic’ application:

- Connects to network with Ethernet (using DHCP)

- Connects to Datavenue Live Objects Plaftorm, using:
  - an optional secure connection (TLS)
  - the LiveObjects mode: Json+Device
- Publishs
  - The current Status/Info
  - The current Configuration Parameters
  - The current Resources
- Subscribes to LiveObjects topics to receive notifications
  - Configuration Parameters update request
  - Resource update request
  - Command request
- then applications waits for an event:
  - From LiveObjects platform to :
    - Update "Configuration Parameters"
    - Update one "Resource" : message or image
    - Process a "Command" : RESET or LED
  - From terminal (through a very simple menu by typing only one character) to perform one of followings:
    - p : Publish message ("Status" message built by user application)
    - d : Push "Collected Data"
    - s : Push "Status"
    - c : Push "Configuration Parameters"
    - r : Push "Resources"
    - R : system reset
  - From application simulating some data publish operations.
  - And if the connection is lost, restart at step 2

3.3.2. Packages dependances

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

1) *mbed-os.lib* (tag: mbed-os-5.2.0 - Promoting release candidate mbed-os-5.2-rc4 to official mbed-os-5.2.0 release)
   `https://github.com/ARMmbed/mbed-os/#e435a07d9252f133ea3d9f6c95dfb176f32ab9b6`
2) *MQTTPacket.lib*
   `https://mbed.org/teams/mqtt/code/MQTTPacket/#62396c1620b6`
3) *jsmn.lib*
   `https://github.com/zserge/jsmn/#1682c32e9ae5990ddd0f0e907270a0f6dde5cbe9`
4) *iotsoftbox-mqtt.lib (the latest release of library on github)*
   `https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-mbed.git/#4360cf49a9dac88274157352b4b8be756f8f2d64`

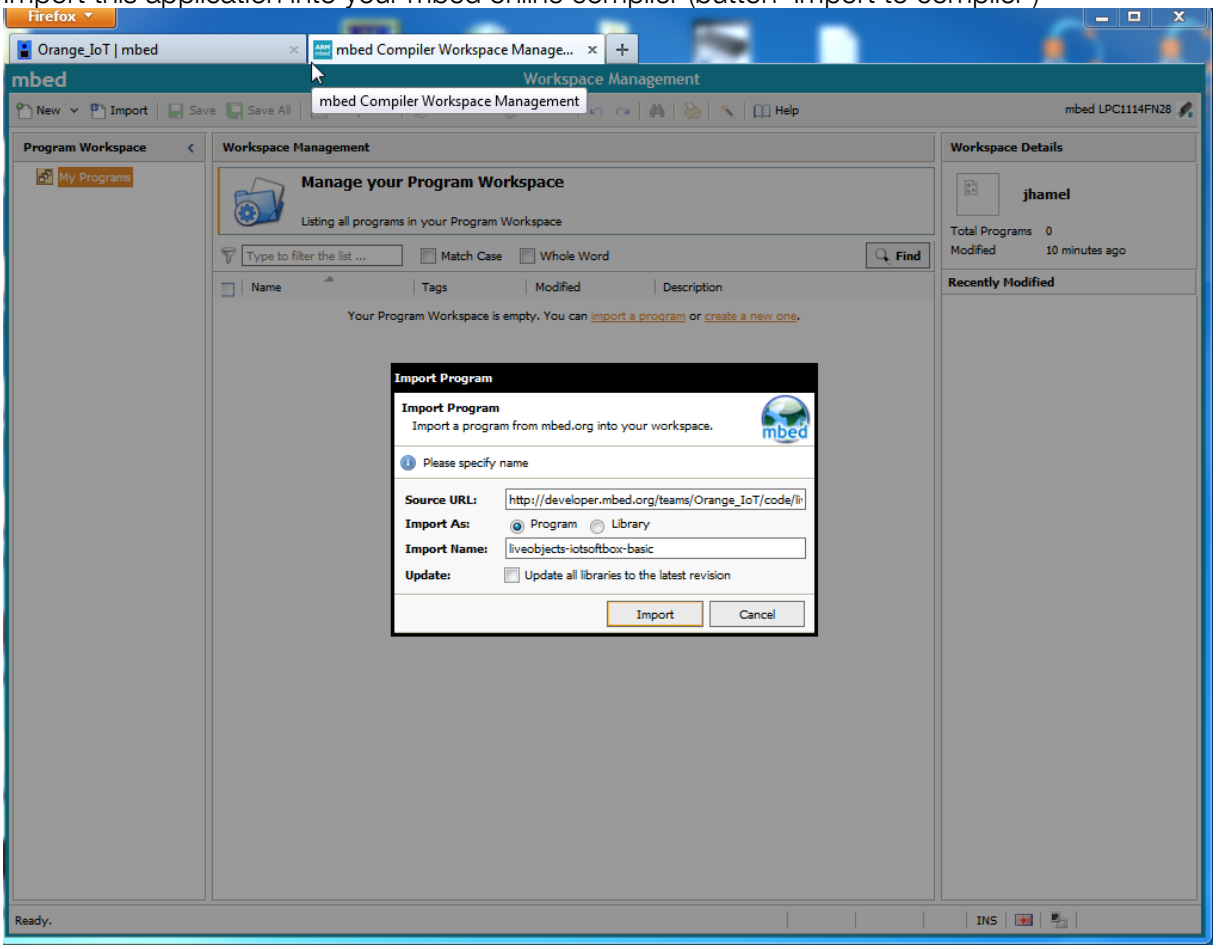
3.3.3. Getting started with mbed online compiler

To help you use mbed online compiler, you can also visit the following page: [getting started with blinky example](..).

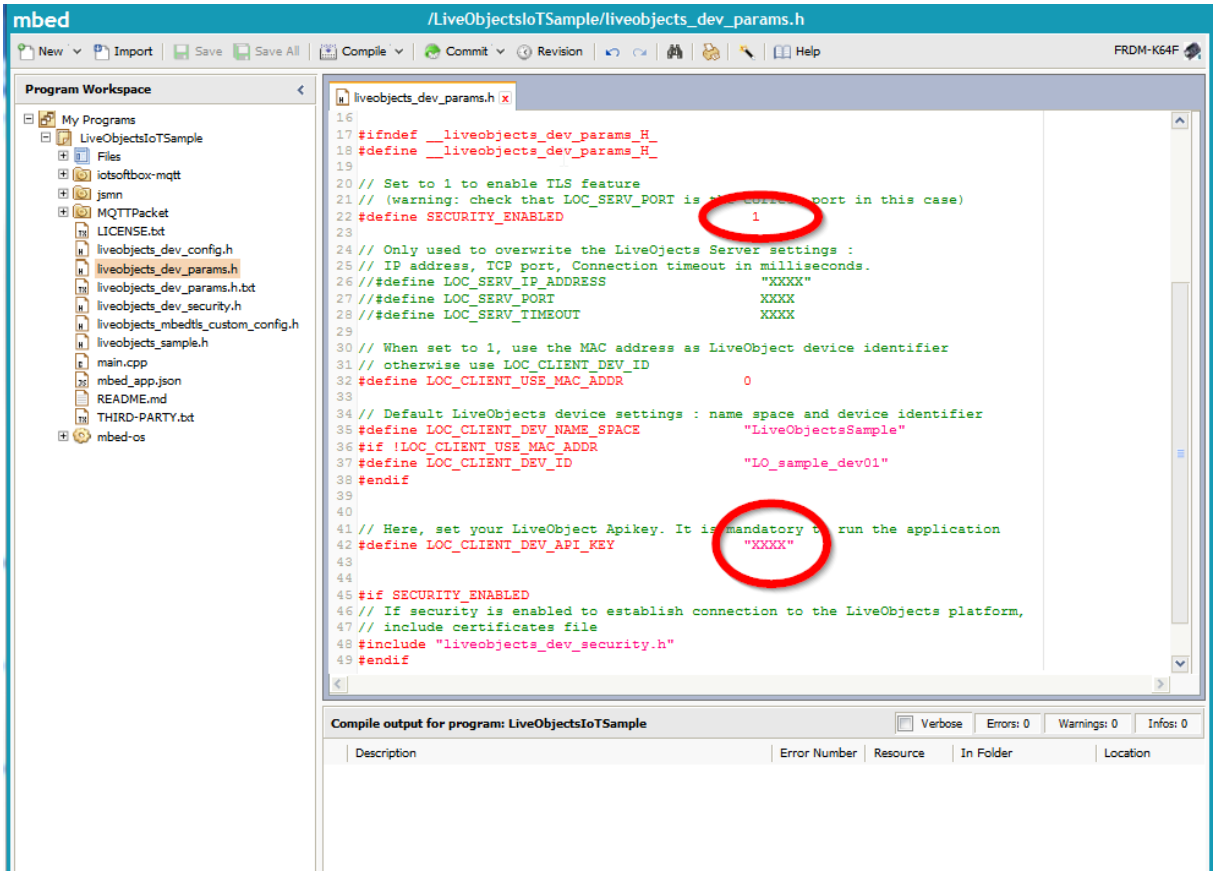
The procedure is the following:

- Join the [mbed OS Developer] community by creating your own [developer account].
- Join the [Orange IoT team]. There are at least two main projects:
  - **liveobjects-iotsoftbox-basic**: A Live Objects 'basic' sample application using this iotsoftbox-mqtt library.
  - **liveobjects-iotsoftbox-greenhouse**: A Live Objects 'greenhouse' demonstrator using this iotsoftbox-mqtt library. The web 'greenhouse' application is located [here].

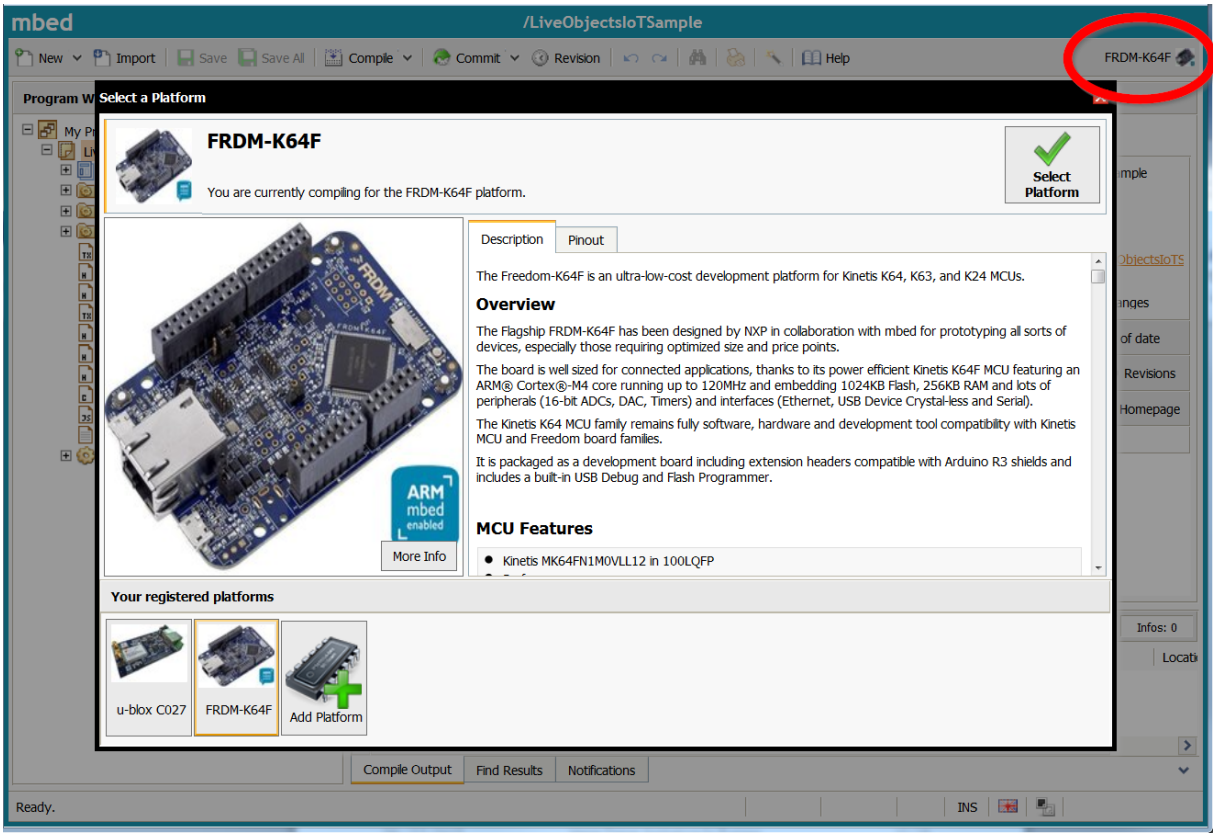
- Select the 'basic' application.
- Import this application into your mbed online compiler (button 'Import to compiler')



- Edit the header file **liveobjects_dev_params.h** to set your Live Objects Tenant API key. And save the header file.



- Select the target FRDM-K64F



- Compile the application for this target
- Download/Save the generated binary file **liveobjects-iotsoftbox-basic_K64F.bin** on your computer.
- Connect your K64F board to your computer with the USB cable. The mbed board should be shown as “mbed removable storage”. And an mbed serial port is up.

  If you met any issue with mbed driver, especially on Windows system, see Windows serial configuration for full details about setting up Windows for serial communication with your mbed Microcontroller.
- Drag and drop the binary file to the board (via the ‘mbed’ storage).
- Start an HyperTerminal application on the mbed serial port (configuration is : 9600 baud, 1-bit stop).
- Connect your K64F board to your network (with a DHCP server).

  (If firewall: outgoing TCP ports are 8883 or 1883).
- Push the reset button on your board.
- The application should be running. And you should see the green LED of your board turning on and off.

3.3.4. Getting starterd with Off-line using mbed CLI

You can also use the mbed Command Line Interface to work off-line (see the mbed tutorial with blinky example).

3.3.4.1. Install tools

- mbed-cli : to build the sample programs. To learn how to build mbed OS applications with mbed-cli, see the user guide.
- GCC ARM Embedded Toolchain: Use 5-2015-q4-major.
- Python 2.7: Use Python 2.7.12 2016-06-25.
- Serial port monitor

Brief install (on a Windows-7 System):

```
# Install TortoiseGit and also Mercurial (TortoiseHg)

# Install Python 2.7.12 (for example in C:\Python27)
# Update your environment variable PATH to add "C:\Python27;C:\Python27\Scripts"

# Install GCC ARM embeded tool chain

# Install mbed-cli
 git clone https://github.com/ARMmbed/mbed-cli
 cd mbed-cli
 python setup.py install

 mbed config --global GCC_ARM_PATH "C:\Program Files (x86)\GNU Tools ARM Embedded\5.2 2015q4\bin"
```

3.3.4.2. Building the sample (using mbed-cli)

1) Clone the **liveobjects-iotsoftbox-basic** repository in a local directory. (Note that the name of this directory will be the name of binary file). To get it, there are two repositories:
   a. On https://developer.mbed.org/teams/Orange_IoT/code/
   b. On https://github.com/Orange-OpenSource/LiveObjects-iotSoftbox-mqtt-mbed-examples containing a collection of examples, in particular the 'basic' sample.
2) Open a command line tool and navigate to the project's directory.
3) Update all sources using the "*mbed update*" command. This command installs packages: mbed-os, MQTTPacket, iotsoftbox-mqtt, and jsmn.
4) Configure the client application:
   a. Edit the header file **liveobjects_dev_params.h** to set your Live Objects Tenant API key.
   b. For others settings, see the specific paragraph: IoT Soft Box Library Configuration.
5) Build the application by selecting the hardware board and build the toolchain using the command "*mbed compile -m K64F -t GCC_ARM*". mbed-cli builds a binary file under the project's *.build* directory.

3.3.4.3. Loading the sample on your K64F board

1) Plug the Ethernet cable into the board if you are using Ethernet mode.
2) Plug the micro-USB cable into the **OpenSDA** port. The board is listed as a mass-storage device.
3) Drag and drop the binary *.build/K64F/GCC_ARM/<local_dir_name>.bin* to the board to flash the application.
4) The board is automatically programmed with the new binary. A flashing LED on it indicates that it is still working. When the LED stops blinking, the board is ready to work.
5) Start the terminal emulator on serial port: mbed Serial Port (COM..).
6) Press the **RESET** button on the board to run the program.

3.3.4.4. Application Monitoring/Testing

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:
    - output: to print debug/trace messages.
    - input: to do some very simple operations by typing only one character. Type 'h' to display the help menu.

4. Detailed Features

4.1. General

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

Today, a library dedicated to the ARM-mbed-OS5 board is available here, library called **LiveObjects-iotSoftbox-mqtt-mbed.**

The **LiveObjects-iotSoftbox-mqtt** library provides APIs to help developers to 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.

The 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:
- TCP Port:
  - 1883 for non SSL connection (without security),
  - 8883 for TSL/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 or not

```
// 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_P1 must be the last sixteen char of the ApiKey
 *
 * If your APIKEY is 0123456789abcdeffedcba9876543210 then
 * it should look like this :
 *
 * #define C_LOC_CLIENT_DEV_API_KEY_P1                  0x0123456789abcdef
 * #define C_LOC_CLIENT_DEV_API_KEY_P2                  0xfedcba9876543210
 *
 * */

#define C_LOC_CLIENT_DEV_API_KEY_P1             0x0123456789abcdef
#define C_LOC_CLIENT_DEV_API_KEY_P2             0xfedcba9876543210
```

When TSL 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 LiveObjects device identifier
// otherwise use LOC_CLIENT_DEV_ID
#define LOC_CLIENT_USE_MAC_ADDR              1

// Default LiveObjects device settings : name space and device identifier
#define LOC_CLIENT_DEV_NAME_SPACE            "LiveObjectsSample"
#if !LOC_CLIENT_USE_MAC_ADDR
#define LOC_CLIENT_DEV_ID                    "LO_sample_dev01"
#endif
```

If `LOC_CLIENT_USE_MAC_ADDR` is set to 1, the softbox-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 boths 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 fom/to the Live Objects platform.

Note that our sample running with mbed OS 5 uses the multi-thread model.

4.5. Status

Status gives information about the device states, i.e. Software version, IP address, GPRS 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:

```
status_handle = LiveObjectsClient_AttachStatus(setOfStatus, SET_STATUS_NB);
```

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

```
static const char* appv_version = "MBED SAMPLE V01.05";
int32_t status_counter = 0;
char    status_message[150] = "READY";

LiveObjectsD_Data_t setOfStatus [] = {
    { LOD_TYPE_STRING_C, "sample_version" ,  (void*)appv_version, 1 },
    { LOD_TYPE_INT32,    "sample_counter" ,  &status_counter, 1 },
    { LOD_TYPE_STRING_C, "sample_message" ,  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):

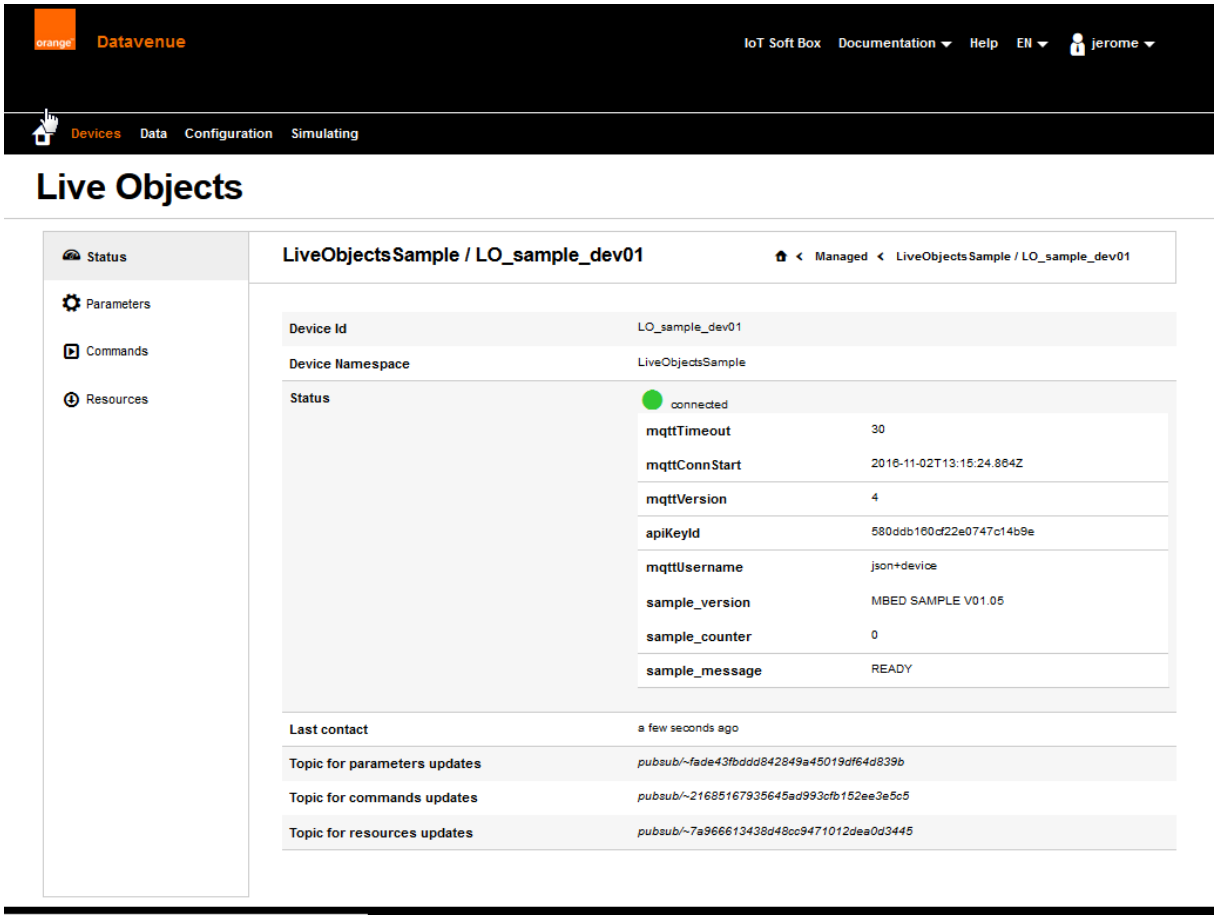
```
ret = LiveObjectsClient_PushStatus ( status_handle );
```

Note:

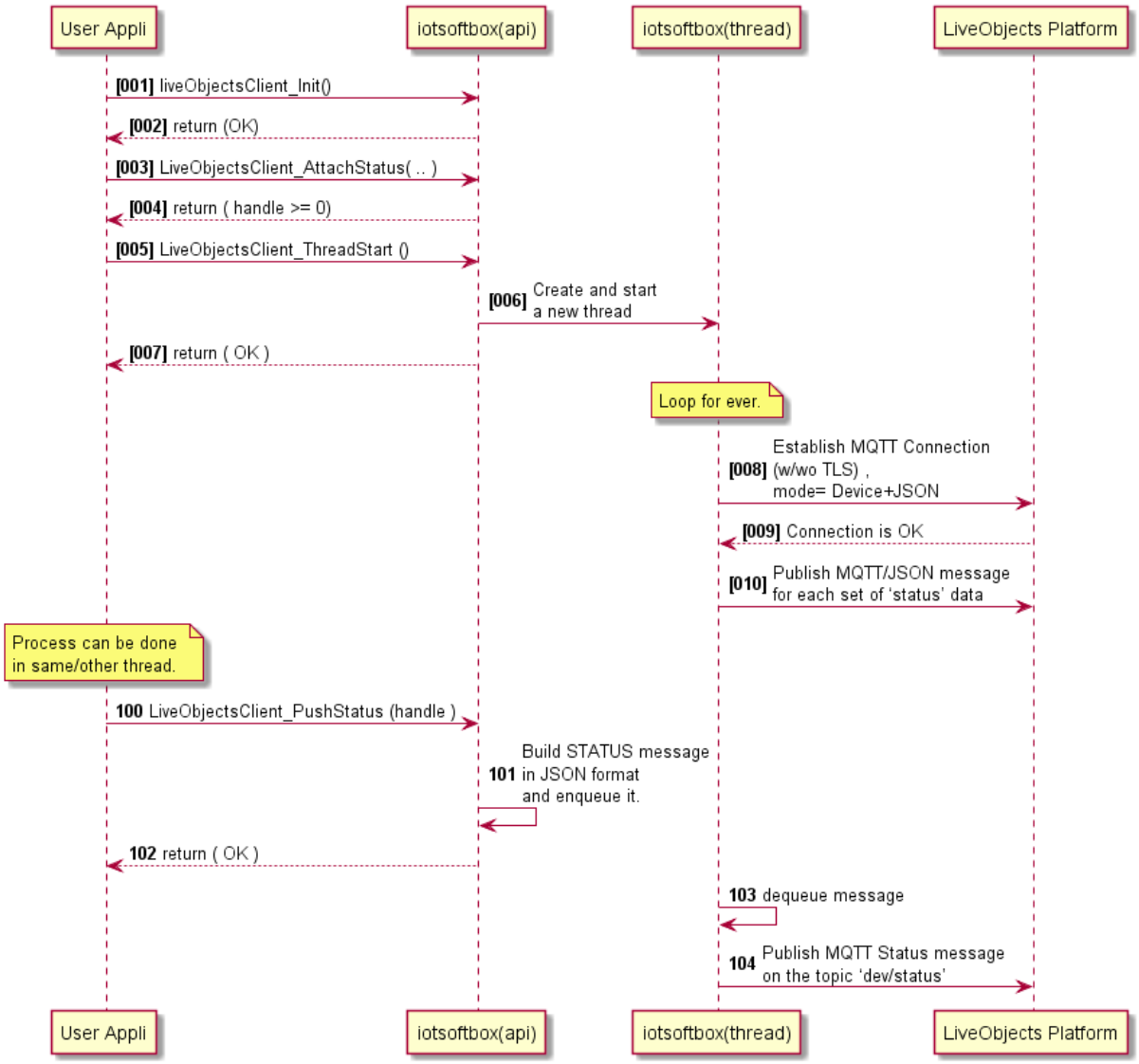
- if the status data is attached before connecting to the platform, the 'status' data will be automatically pushed as soon as the MQTT connection is established with the Live Objects platform.

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:



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 try to 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_AttachParameters (setOfParam, SET_PARAM_NB, paramUdpdateCb);
```

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 setOfParam[] = {
    { 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(setOfParam) / sizeof(LiveObjectsD_Param_t))
```

And the configuration parameters are defined and initialized as:

```
volatile uint32_t appv_cfg_timeout = 10;

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

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

```
extern "C" int paramUdpdateCb (
        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:
        ...
        break;
    case PARM_IDX_TIMEOUT:
        ...
          return 0; // primitive parameter is updated by library
        break;
    case PARM_IDX_THRESHOLD:
        ...
        break;
    case PARM_IDX_GAIN:
         ...
        break;
    }
    return -1;
}
```

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.

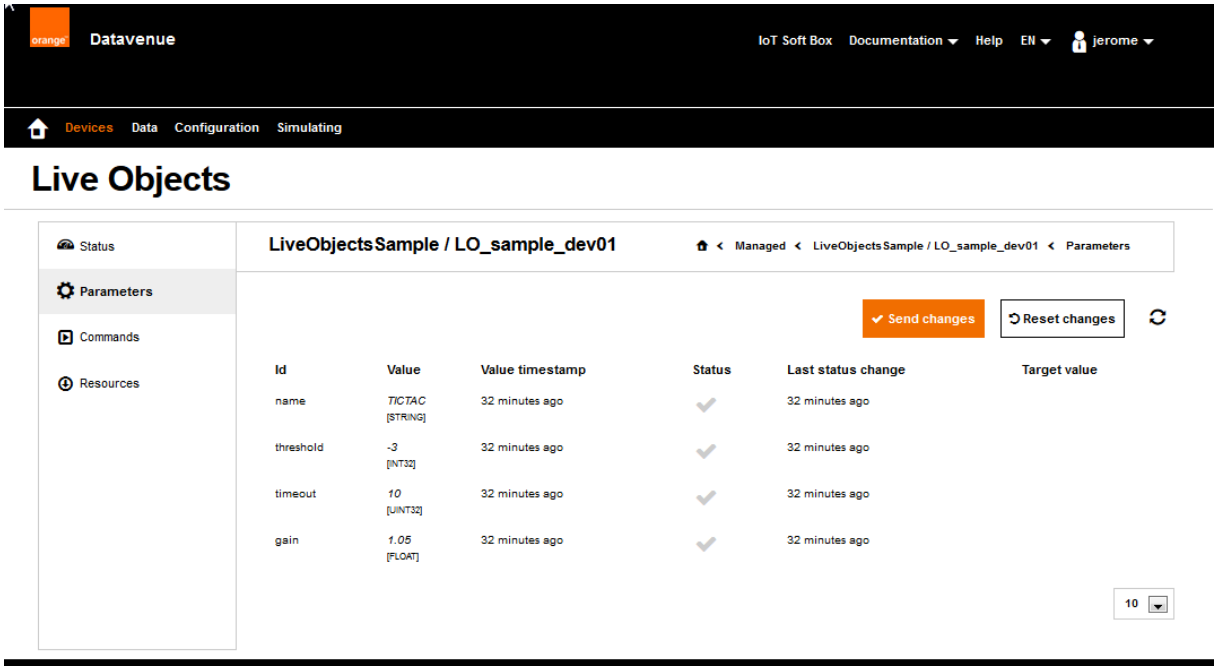
4.6.2. Push a set of configuration parameters

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 cuurent configuration is updated:

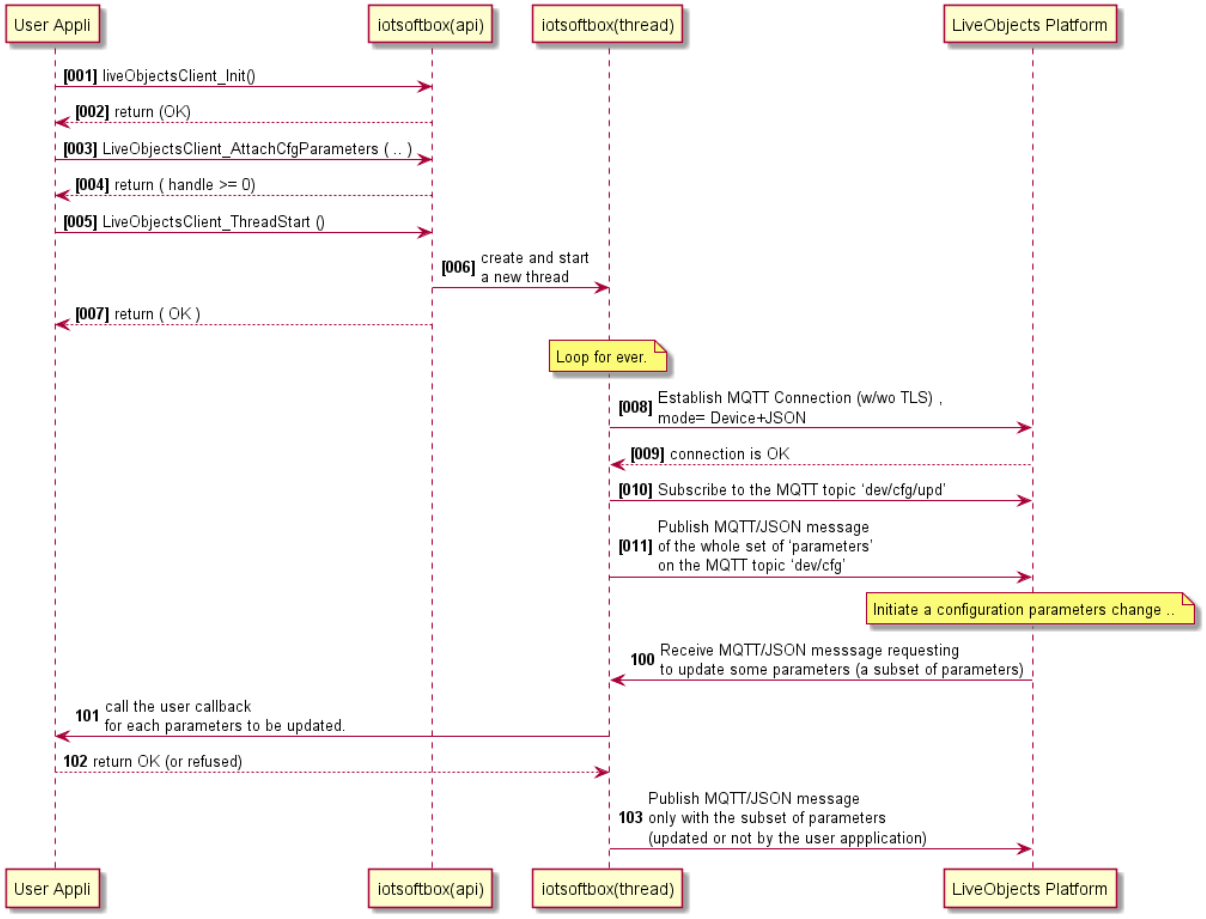
```
ret = LiveObjectsClient_PushCfgParams ( );
```

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:



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:

```
hdl_data = LiveObjectsClient_AttachData(STREAM_PREFIX,
      "LO_sample_measures",
      "mVl","\"Test\"", NULL,
      setOfMeasures, SET_MEASURES_NB);
```

Where:

```
/// Set of Collected data (published on a data stream)
LiveObjectsD_Data_t setOfMeasures[] = {
    { LOD_TYPE_UINT32, "counter" ,        &measures_counter, 1 },
    { LOD_TYPE_INT32,  "temperature" ,    &measures_temp, 1 },
    { LOD_TYPE_FLOAT,  "battery_level" ,  &measures_volt, 1 }
};
#define SET_MEASURES_NB (sizeof(setOfMeasures) / sizeof(LiveObjectsD_Data_t))
```

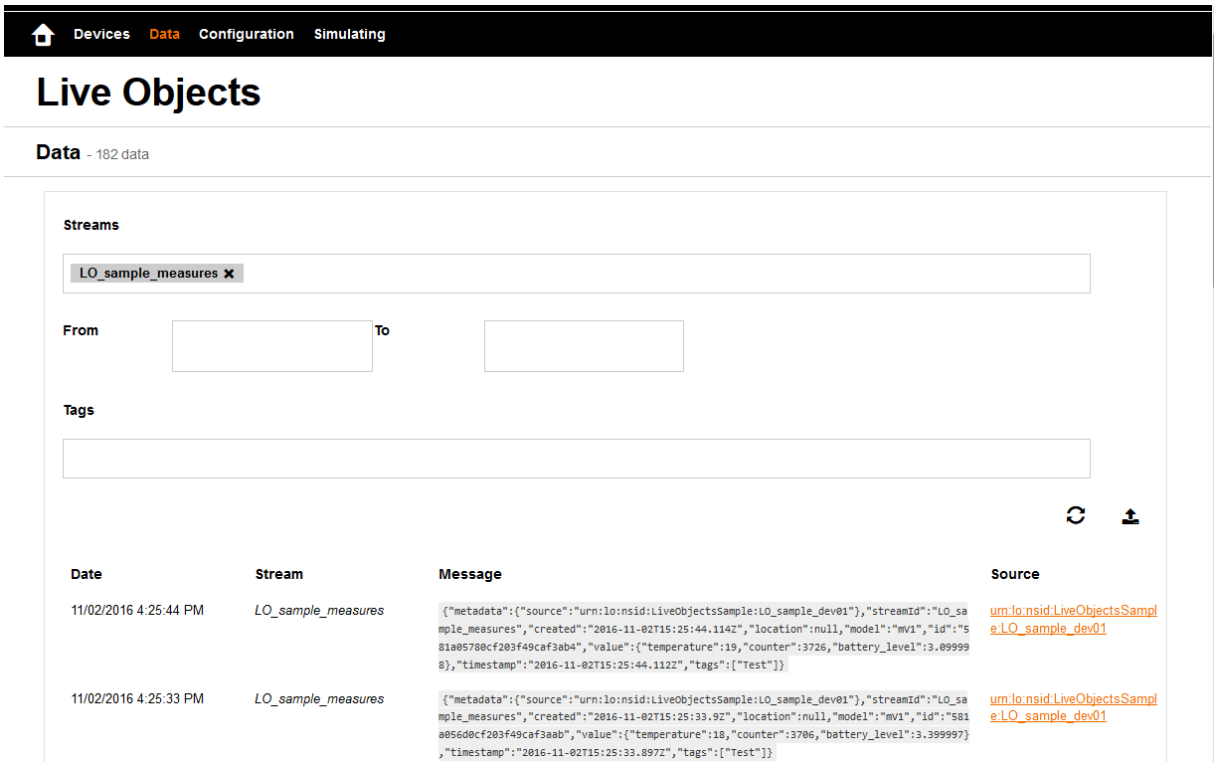
4.7.2. Push the set of collected data

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):

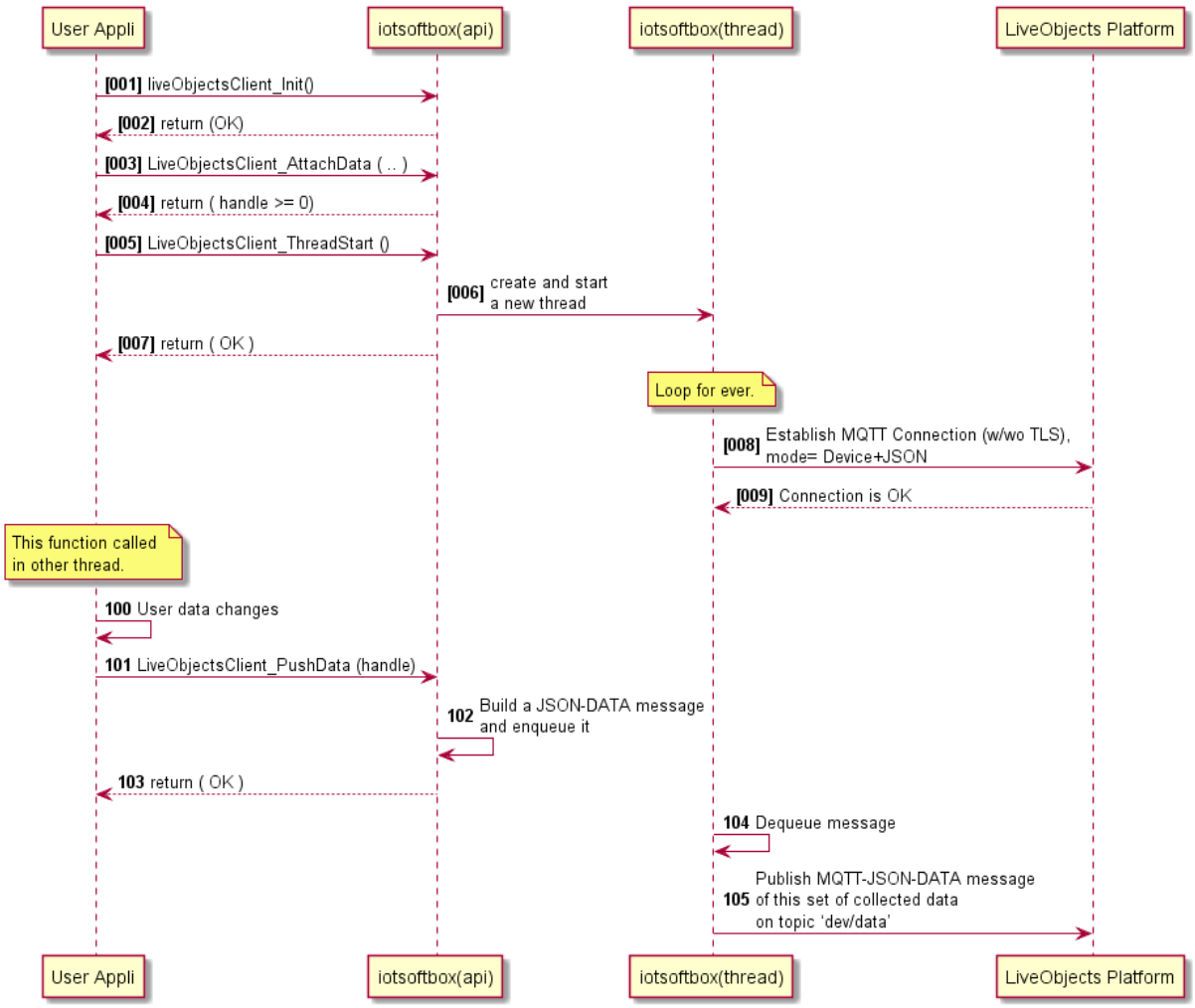
```
ret = LiveObjectsClient_PushData ( hdl_data );
```

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):



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(setOfCommands, SET_COMMANDS_NB, commandCb);
```

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

/// set of commands
LiveObjectsD_Command_t setofCommands[] = {
    { CMD_IDX_RESET, "RESET" , 0},
    { CMD_IDX_LED,   "LED"   , 0}
};
#define SET_COMMANDS_NB (sizeof(setofCommands) / 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.

```
/// Called (by the LiveObjects thread) to perform an 'attached/registered' command
extern "C" int commandCb(LiveObjectsD_CommandRequestBlock_t* pCmdReqBlk)
{
    int ret;
    const LiveObjectsD_Command_t*  cmd_ptr;

    ...  // maybe check input parameters

    cmd_ptr = pCmdReqBlk->hd.cmd_ptr;

    switch(cmd_ptr->cmd_uref) {
    case CMD_IDX_RESET: // RESET
        ...
        ret = 1:   // result is OK
        break;

    case CMD_IDX_LED: // LED
        ...
        ret = 0;   // pending
        break;
    default :
        ret = -4;
    }
    return ret;
}
```

4.8.2. Enable/disable 'command' feature

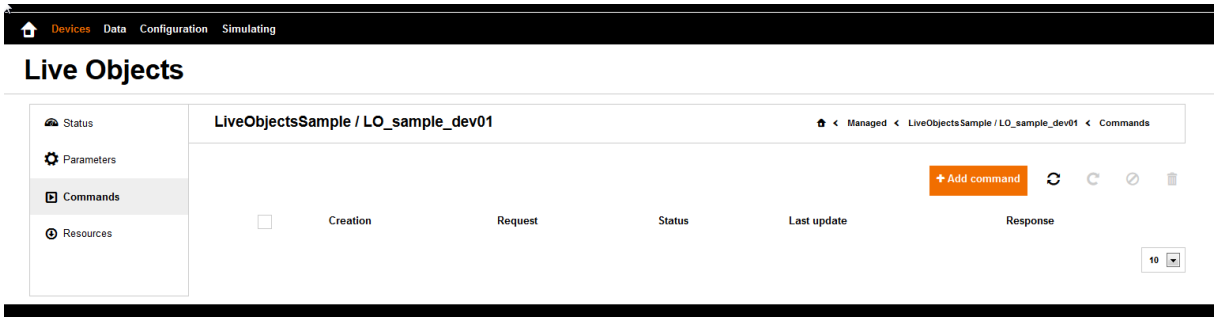
As soon as the device is ready (or not) to process commands, the application can enable (or disable) the 'command' feature by calling the function:

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

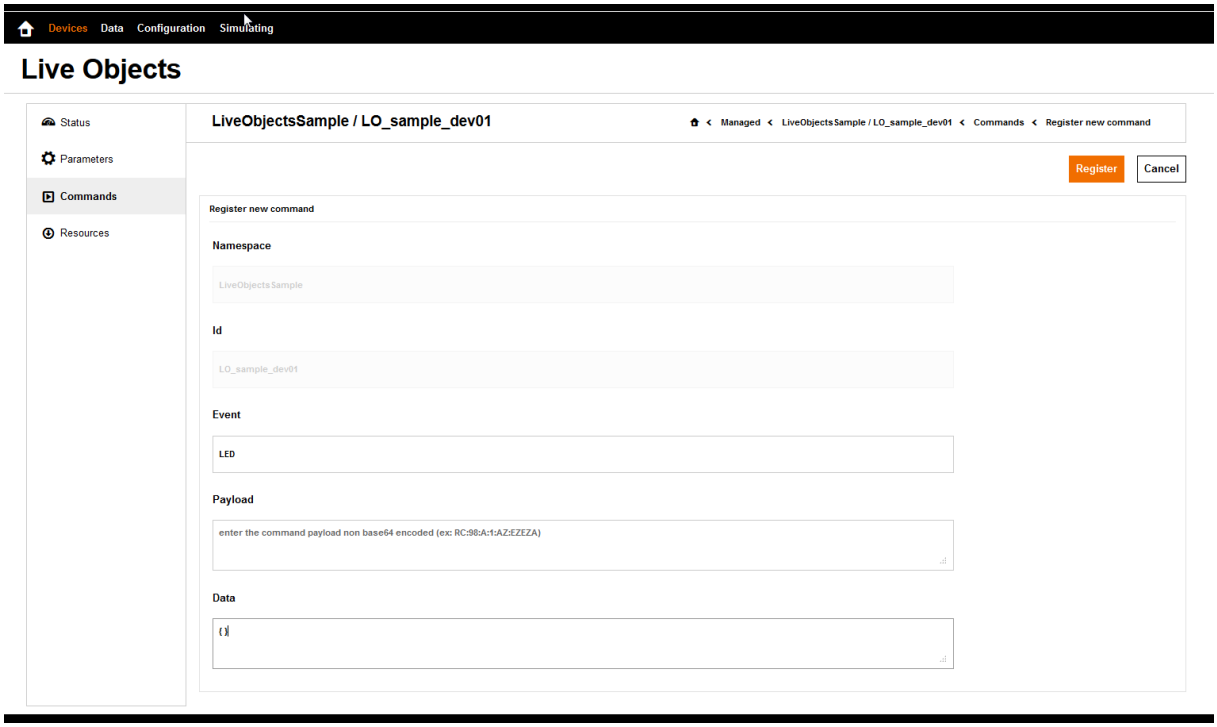
4.8.3. Use of Live Objects Portal to send a command

On the Live Objects Portal,

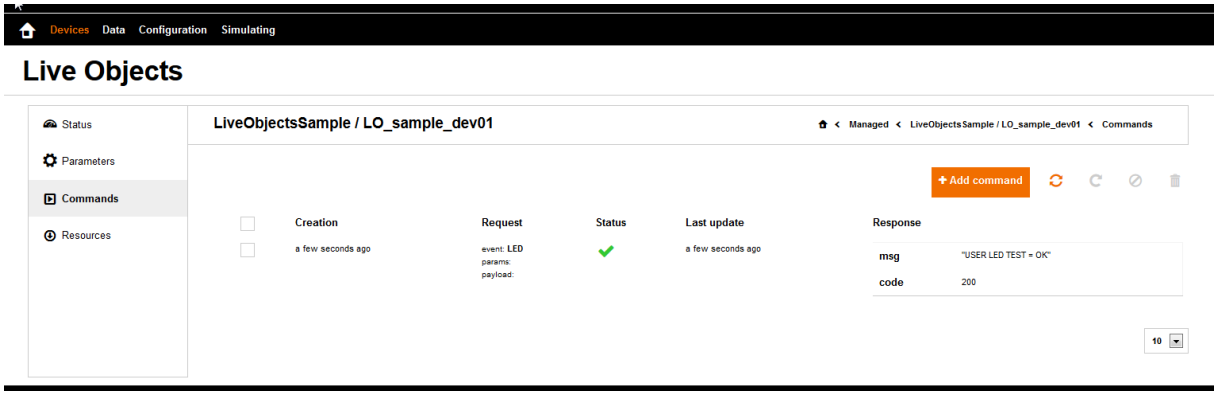
- Go to tab : Devices -> <your device> -> Commands



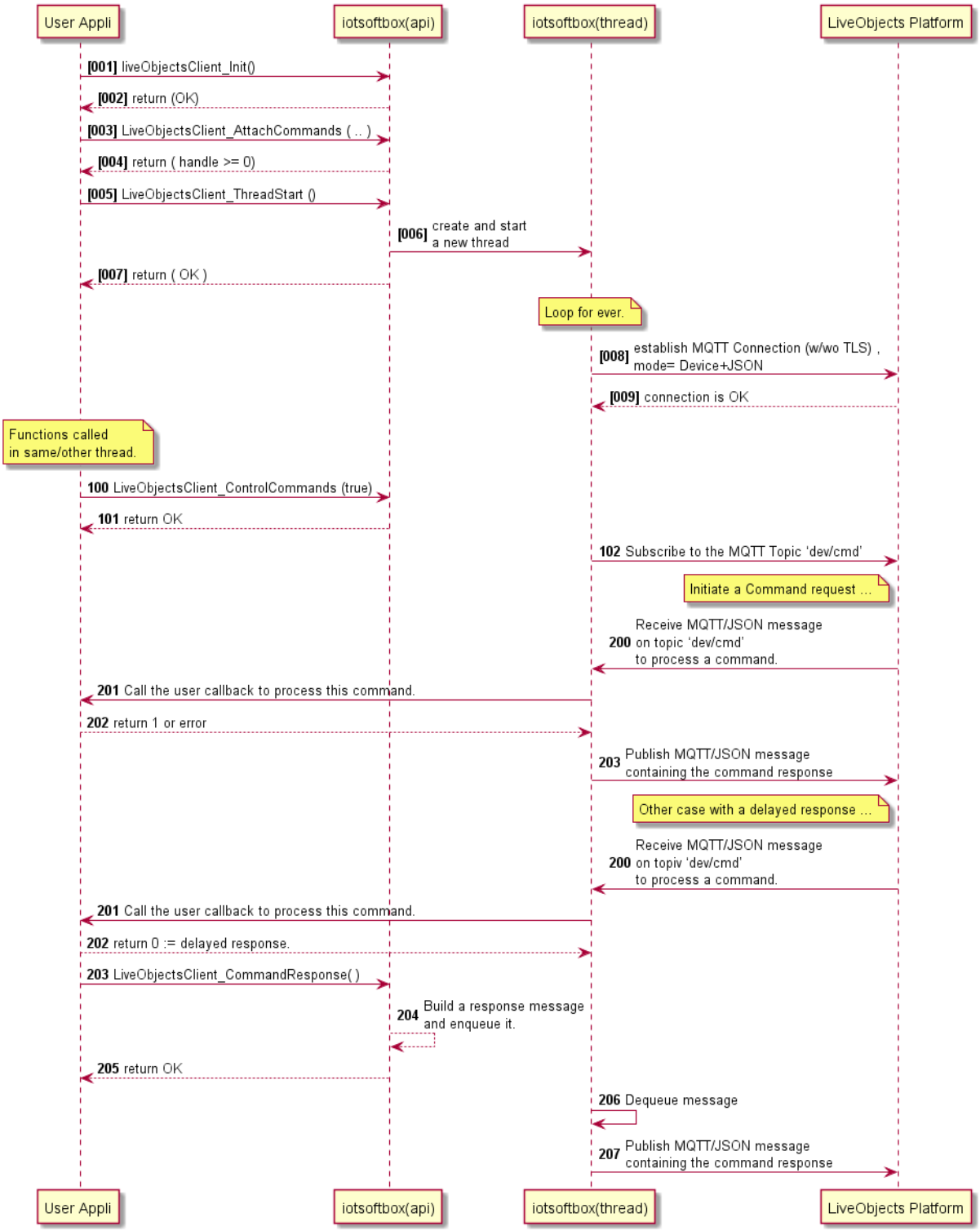
- Click on button '+ Add command"



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



4.8.4. Sequence Diagram


User Appli
iotsoftbox(api)
iotsoftbox(thread)
LiveObjects Platform
[001] liveObjectsClient_Init()
[002] return (OK)
[003] LiveObjectsClient_AttachCommands (..)
[004] return (handle >= 0)
[005] LiveObjectsClient_ThreadStart ()
[006] create and start a new thread
[007] return (OK)
Loop for ever.
[008] establish MQTT Connection (w/wo TLS) , mode= Device+JSON
[009] connection is OK
Functions called in same/other thread.
100 LiveObjectsClient_ControlCommands (true)
101 return OK
102 Subscribe to the MQTT Topic 'dev/cmd'
Initiate a Command request ...
200 Receive MQTT/JSON message on topic 'dev/cmd' to process a command.
201 Call the user callback to process this command.
202 return 1 or error
203 Publish MQTT/JSON message containing the command response
Other case with a delayed response ...
200 Receive MQTT/JSON message on topiv 'dev/cmd' to process a command.
201 Call the user callback to process this command.
202 return 0 := delayed response.
203 LiveObjectsClient_CommandResponse( )
204 Build a response message and enqueue it.
205 return OK
206 Dequeue message
207 Publish MQTT/JSON message containing the command response
User Appli
iotsoftbox(api)
iotsoftbox(thread)
LiveObjects Platform

4.9. Resources

4.9.1. Attach a set of resources

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

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

In the sample application:

```
ret = LiveObjectsClient_AttachResources(setOfResources, SET_RESOURCES_NB,
           rscNtfyCb, rscDataCb);
```

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 setOfresources[] = {
    { 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(setOfresources) / 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:

```
/**
 * Called (by the LiveObjects library) to notify either,
 *  - state = 0  : the begin of resource request
 *  - state = 1  : the end without error
 *  - state != 1 : the end with an error
 */
extern "C"  LiveObjectsD_ResourceRespCode_t  rscNtfyCb (
        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) { // Completed
            if (state == 1) { // Completed without error
               ...
            }
            else { // Completed with error
               ...
            }
        }
        else { // Started
            ret = RSC_RSP_ERR_NOT_AUTHORIZED;
            switch (rsc_ptr->rsc_uref ) {
                case RSC_IDX_MESSAGE:
                    ...
                    ret = RSC_RSP_OK;
                    break;
                case RSC_IDX_IMAGE:
                    ...
                    ret = RSC_RSP_OK;
                   break;
            }
            if (ret == RSC_RSP_OK) {
                // Initialize the transfer
               ...
            }
            else { // Transfer is refused
               ...
            }
        }
    }
    else {
        ret = RSC_RSP_ERR_INVALID_RESOURCE;
    }
    return ret;
}
```

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

   In the sample application:

```
/**
 * Called (by the LiveObjects library) to request the user
 * to read data from current resource transfer.
 */
extern "C"  int rscDataCb (const LiveObjectsD_Resource_t* rsc_ptr, uint32_t offset)
{
    int ret;

    if (rsc_ptr->rsc_uref == RSC_IDX_IMAGE) {
        if (offset > (sizeof(rsc_image)-1)) {
             return -1;
         }
        int data_len = sizeof(rsc_image) - offset - 1;
        ret = LiveObjectsClient_RscGetChunck(rsc_ptr, &rsc_image[offset], data_len);
        if (ret > 0) {
            if ((offset+ret) > (sizeof(rsc_image)-1)) {
                 return -1;
            }
            ...
        }
    }
    else if ( .. ) {
```

```
        ...
    }
    else {
        ret = -1;
    }
    return ret;
}
```

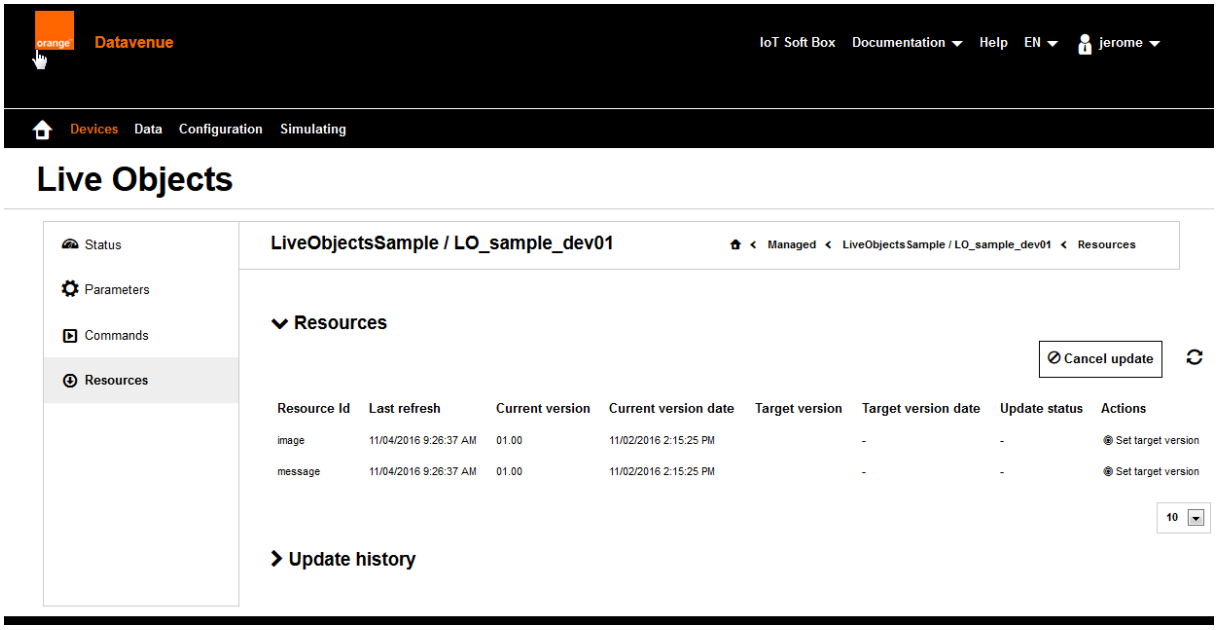
4.9.2. Enable/disable 'resources' feature

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 );
```

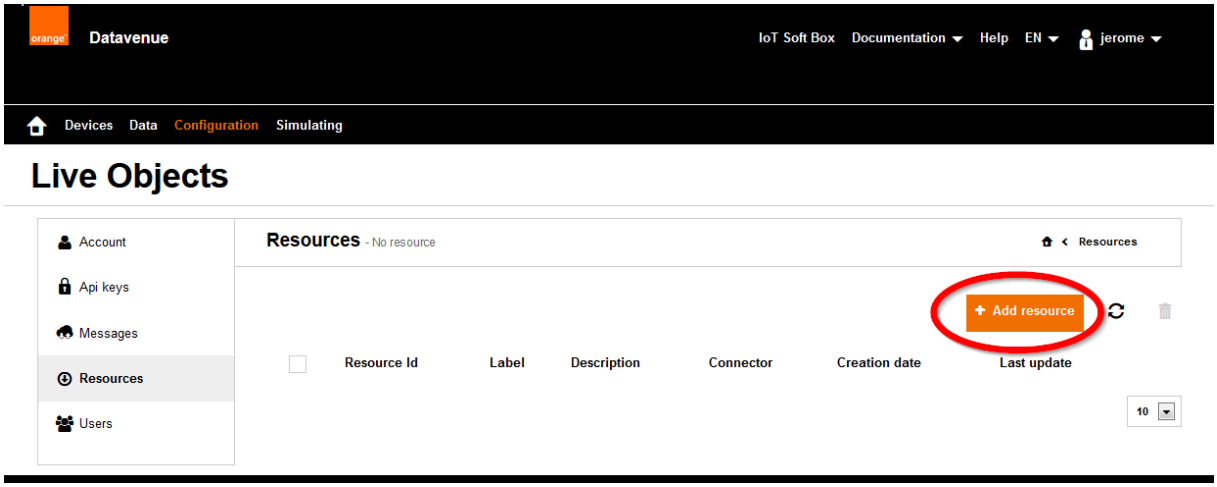
4.9.3. Use of Live Objects Portal to create and update a resource

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



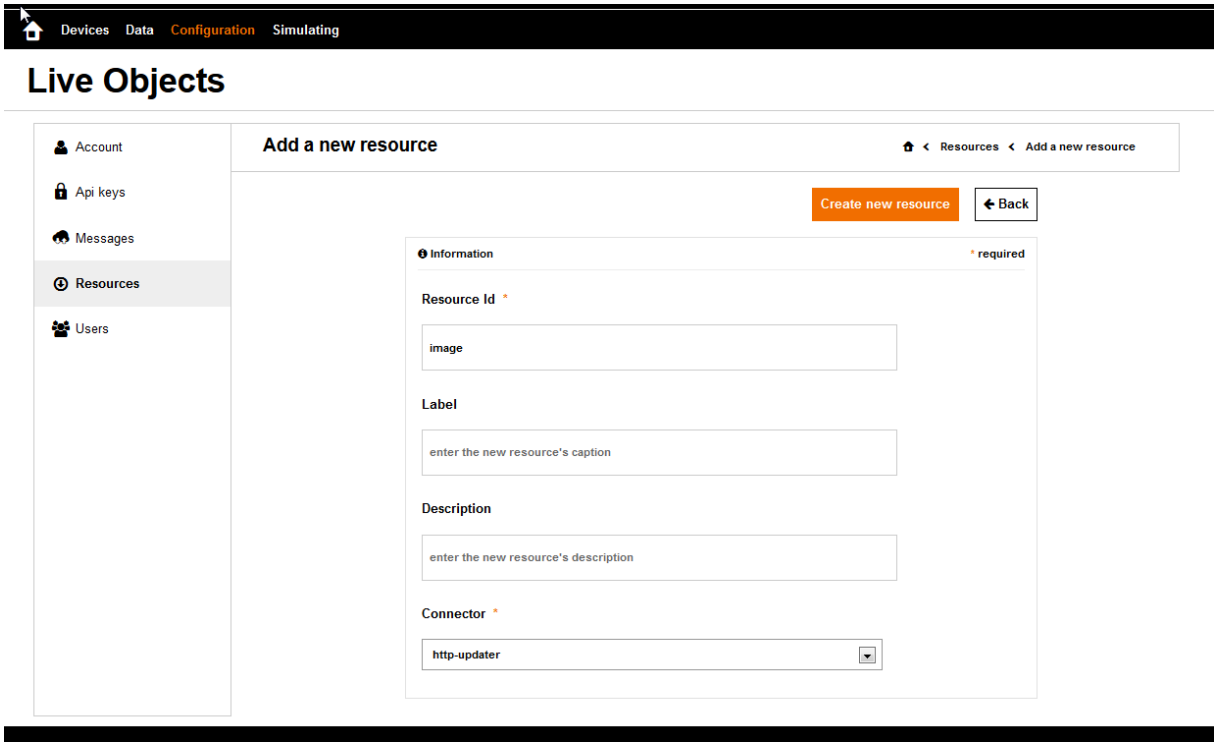
Here, the device 'LiveObjectsSample/LO_sample_dev01' has two resources identified by: *image* and *message*.

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

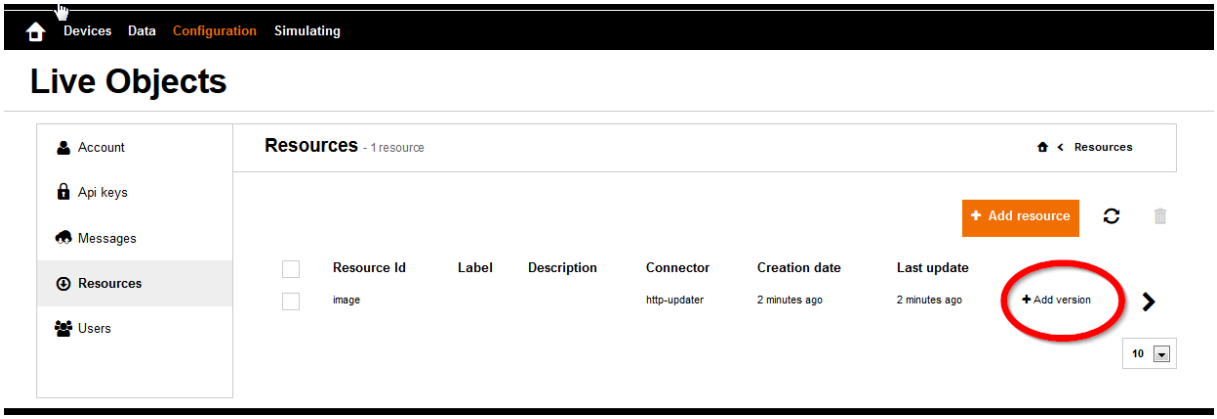


Two fields are mandatory

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

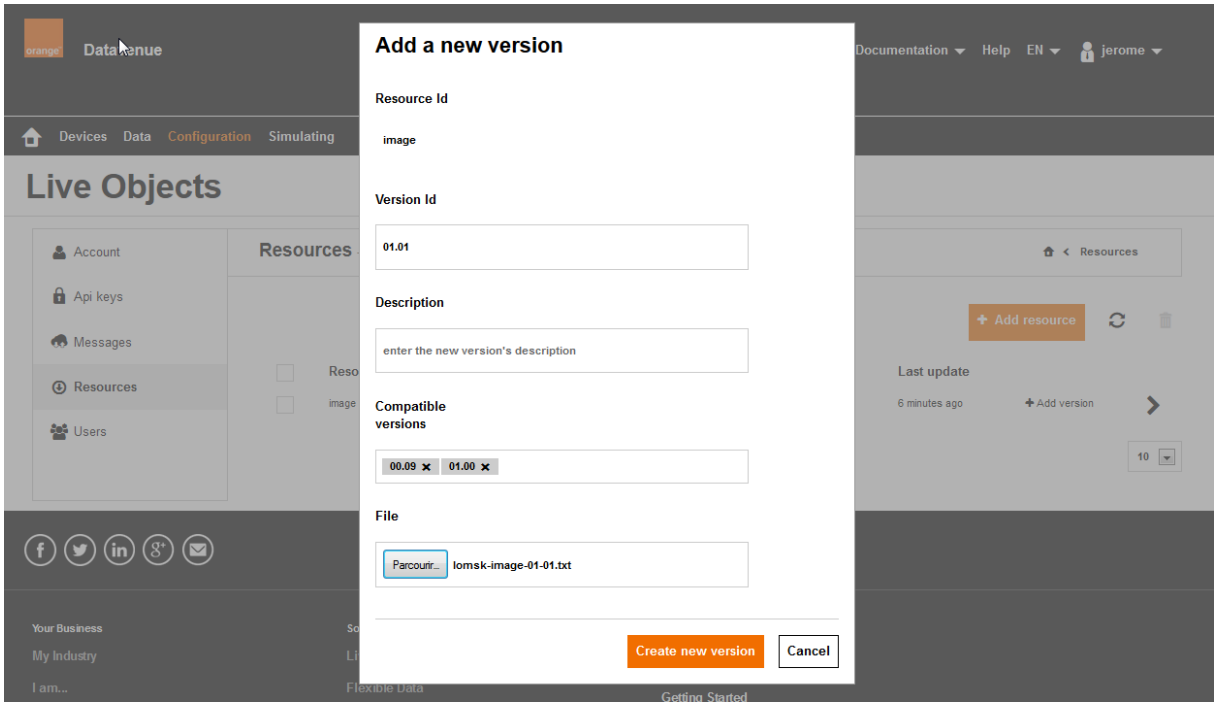


The result is the following:



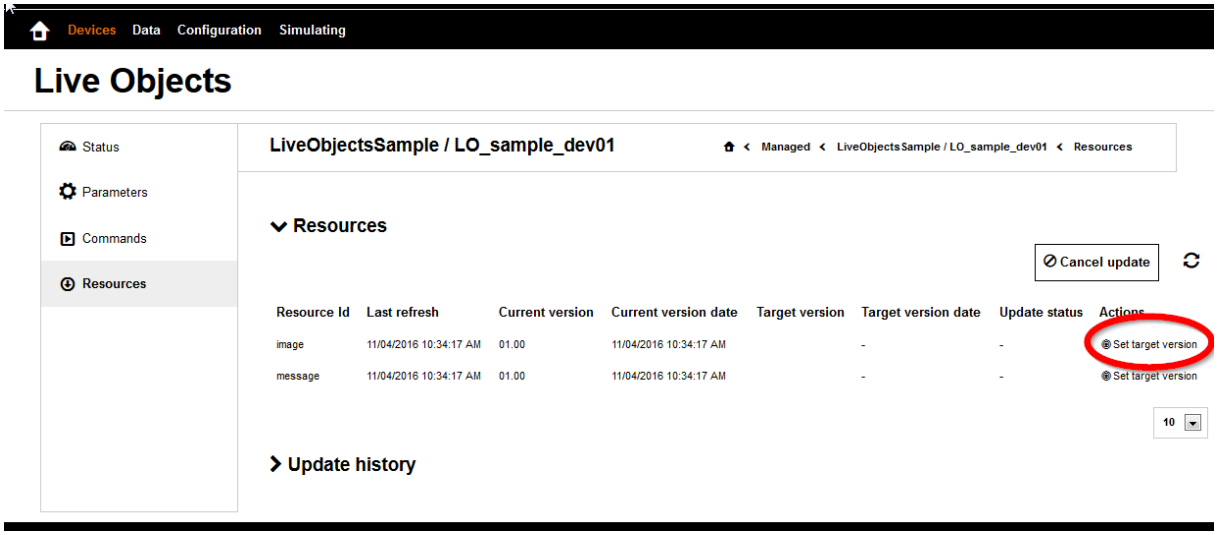
Then, a new resource version can be attached to this resource 'image', by specifying:

- 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

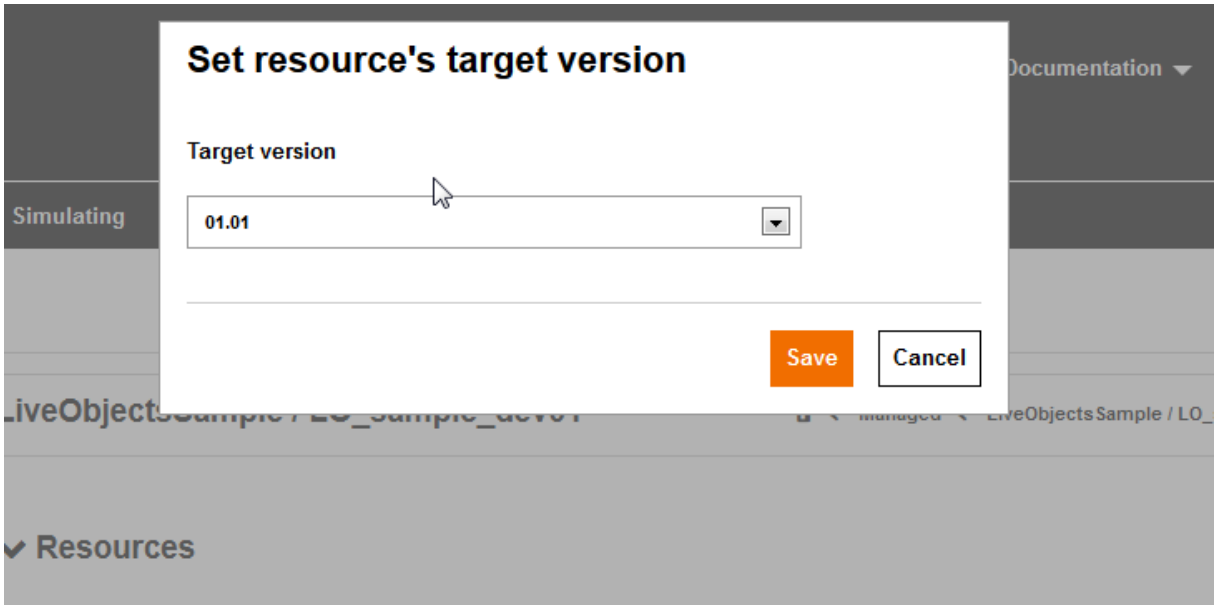


Now, the resource update request can be launched for this device:

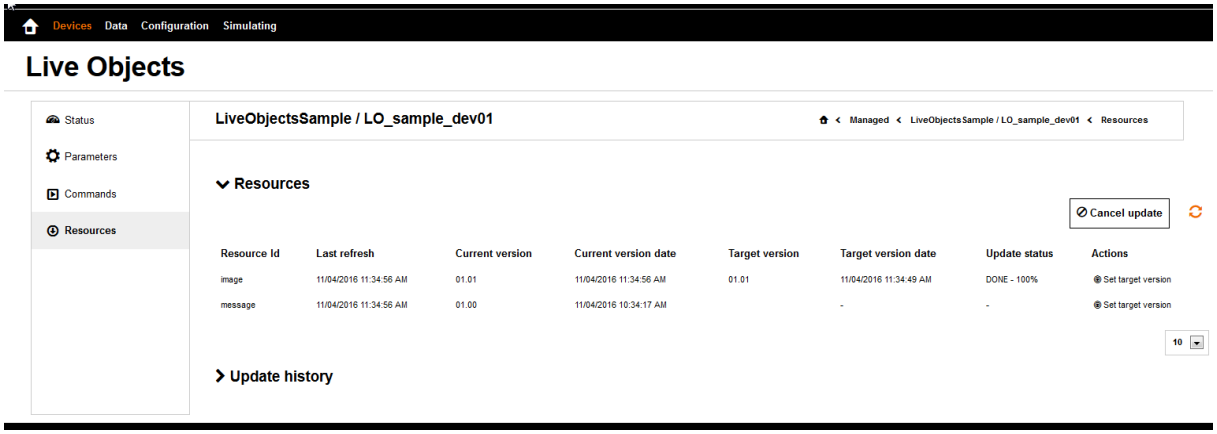
- Go to the 'devices' tab.
- Select your connected device, here it is "LiveObjectsSample / LO_sample_dev01"
- Go to the 'resources' tab



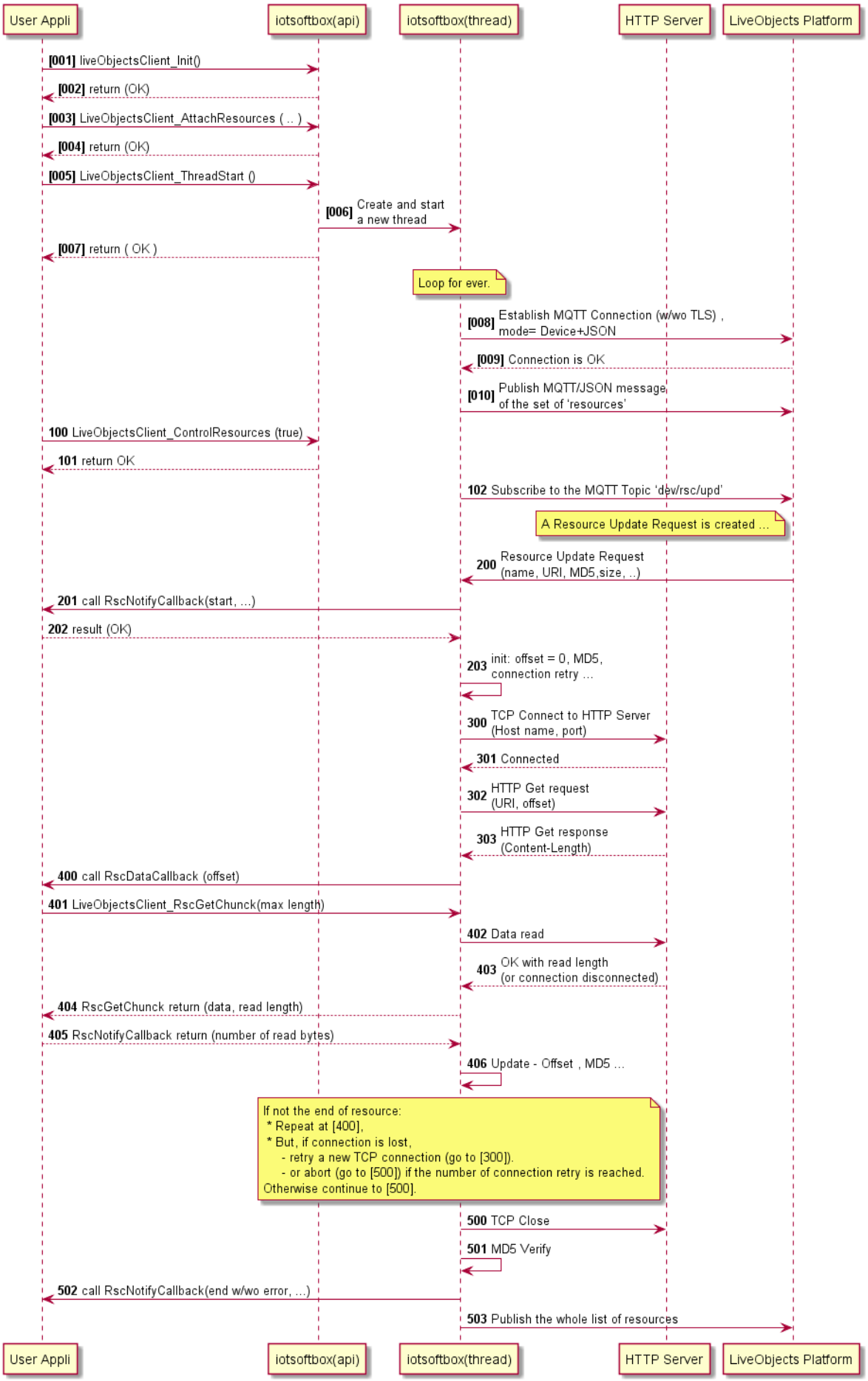
- Click on 'Set target version'
- And select the resource version to download on device

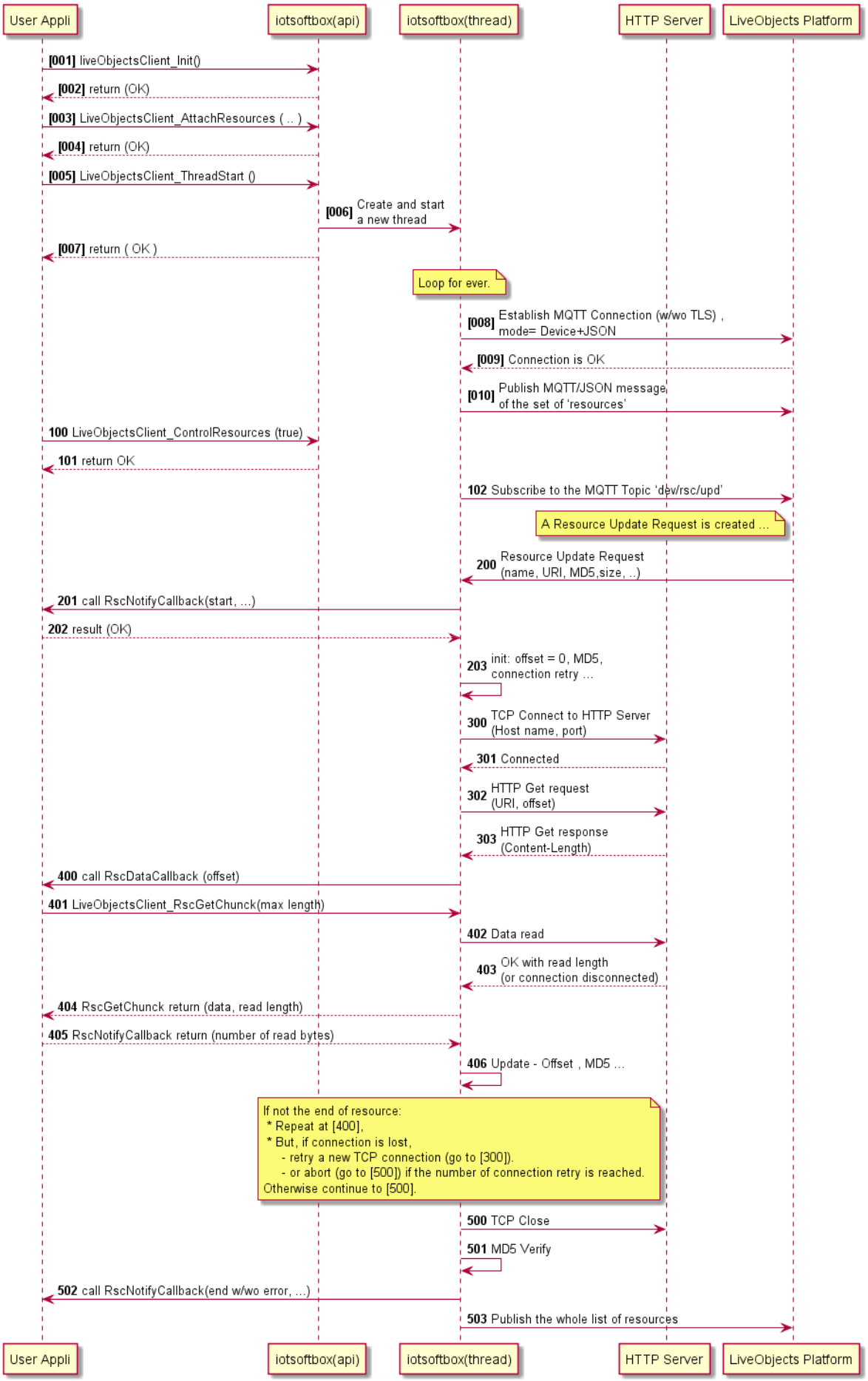


At the end of transfer, after refreshing the web page, the current version should be equal to the target version:



4.9.4. Sequence diagram


User Appli
iotsoftbox(api)
iotsoftbox(thread)
HTTP Server
LiveObjects Platform
[001] liveObjectsClient_Init()
[002] return (OK)
[003] LiveObjectsClient_AttachResources (..)
[004] return (OK)
[005] LiveObjectsClient_ThreadStart ()
[006] Create and start a new thread
[007] return (OK)
Loop for ever.
[008] Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
[009] Connection is OK
[010] Publish MQTT/JSON message of the set of 'resources'
100 LiveObjectsClient_ControlResources (true)
101 return OK
102 Subscribe to the MQTT Topic 'dev/rsc/upd'
A Resource Update Request is created ...
200 Resource Update Request (name, URI, MD5,size, ..)
201 call RscNotifyCallback(start, ...)
202 result (OK)
203 init: offset = 0, MD5, connection retry ...
300 TCP Connect to HTTP Server (Host name, port)
301 Connected
302 HTTP Get request (URI, offset)
303 HTTP Get response (Content-Length)
400 call RscDataCallback (offset)
401 LiveObjectsClient_RscGetChunck(max length)
402 Data read
403 OK with read length (or connection disconnected)
404 RscGetChunck return (data, read length)
405 RscNotifyCallback return (number of read bytes)
406 Update - Offset , MD5 ...
If not the end of resource:
* Repeat at [400],
* But, if connection is lost,
- retry a new TCP connection (go to [300]).
- or abort (go to [500]) if the number of connection retry is reached.
Otherwise continue to [500].
500 TCP Close
501 MD5 Verify
502 call RscNotifyCallback(end w/wo error, ...)
503 Publish the whole list of resources
User Appli
iotsoftbox(api)
iotsoftbox(thread)
HTTP Server
LiveObjects Platform


User Appli
iotsoftbox(api)
iotsoftbox(thread)
HTTP Server
LiveObjects Platform
[001] liveObjectsClient_Init()
[002] return (OK)
[003] LiveObjectsClient_AttachResources (..)
[004] return (OK)
[005] LiveObjectsClient_ThreadStart ()
[006] Create and start a new thread
[007] return (OK)
Loop for ever.
[008] Establish MQTT Connection (w/wo TLS) , mode= Device+JSON
[009] Connection is OK
[010] Publish MQTT/JSON message of the set of 'resources'
100 LiveObjectsClient_ControlResources (true)
101 return OK
102 Subscribe to the MQTT Topic 'dev/rsc/upd'
A Resource Update Request is created ...
200 Resource Update Request (name, URI, MD5,size, ..)
201 call RscNotifyCallback(start, ...)
202 result (OK)
203 init: offset = 0, MD5, connection retry ...
300 TCP Connect to HTTP Server (Host name, port)
301 Connected
302 HTTP Get request (URI, offset)
303 HTTP Get response (Content-Length)
400 call RscDataCallback (offset)
401 LiveObjectsClient_RscGetChunck(max length)
402 Data read
403 OK with read length (or connection disconnected)
404 RscGetChunck return (data, read length)
405 RscNotifyCallback return (number of read bytes)
406 Update - Offset , MD5 ...
If not the end of resource:
* Repeat at [400],
* But, if connection is lost,
- retry a new TCP connection (go to [300]).
- or abort (go to [500]) if the number of connection retry is reached.
Otherwise continue to [500].
500 TCP Close
501 MD5 Verify
502 call RscNotifyCallback(end w/wo error, ...)
503 Publish the whole list of resources
User Appli
iotsoftbox(api)
iotsoftbox(thread)
HTTP Server
LiveObjects Platform

An example of URI is:
http://liveobjects.orange-business.com:80/dl/18p1bj775jhk0pj6p49076hk45

And the header of HTTP Get Response is something like that:

```
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
```

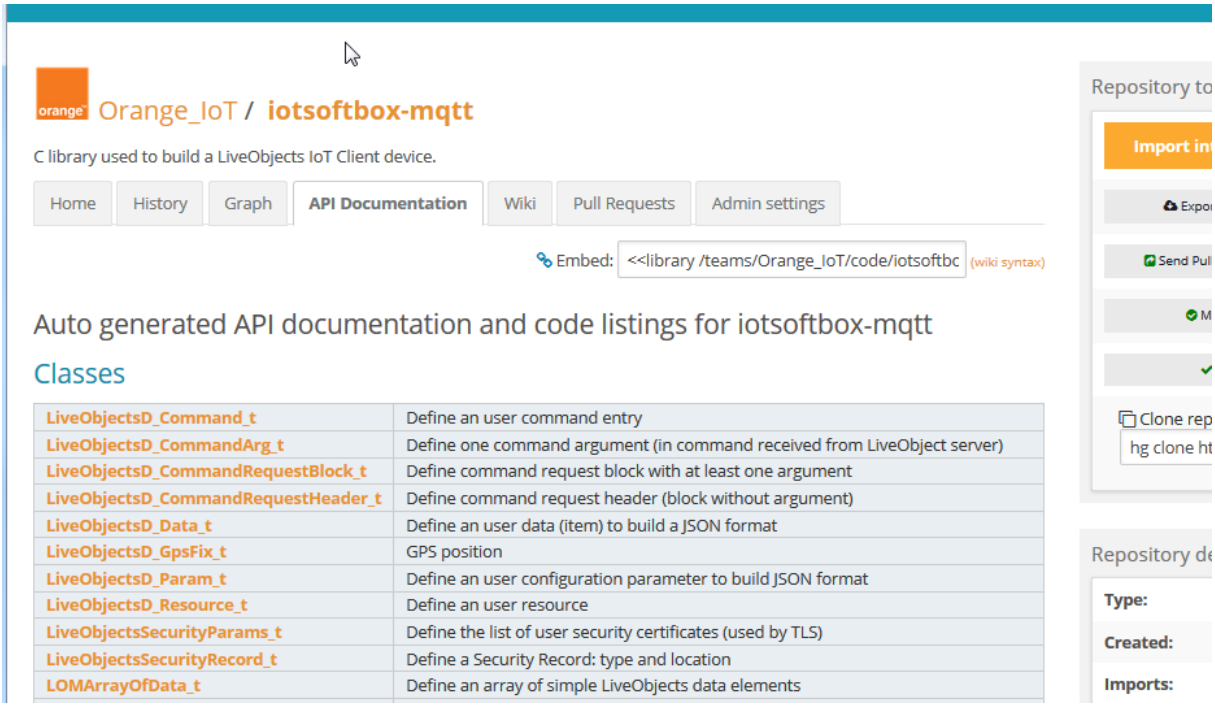
5. Additionnal Information

5.1. Doxygen documentation

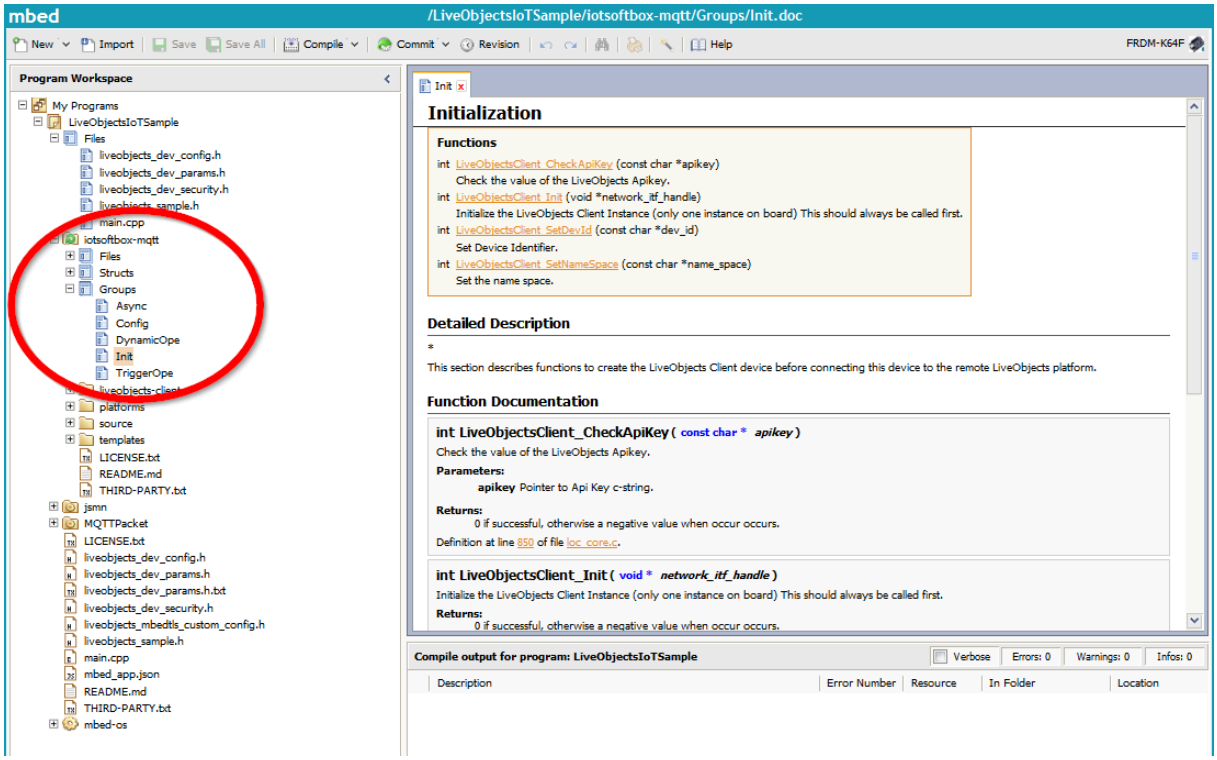
The iotsoftbox-mqtt library is documented using the Doxygen source code comments (mainly for the '*public*' header files).

5.1.1. ARM mbed environment

- Auto generated API documentation on the developer.mbed.org site.



- Mbed online compiler



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. ARM mbed environment

The `LOTRACE_xxx` MACROs are mapped onto mbed trace functions (defined in mbed-trace/mbed_trace.h).

If necessary, trace feature can be also enabled in the MQTTPacket library. A template file *StackTrace.h.txt* is provided to replace the header file *StackTrace.h* in the MQTTPacket directory. Functions are implemented in *MQTTLog.c* file

And the following trace goups are defined:

- LOC : Live Objects Client
- JSON : Encode JSON messages
- JMSG : Decode JSON message (using JSM library)
- MQTT : MQTT Packet Library

Then the user application has to setup the mbed trace feature.

Here a sample code to enable mbed trace feature in multi-thread application:

```
Mutex trace_mutex;

extern "C"  void trace_mutex_wait(void) {
    trace_mutex.lock();
}

extern "C"  void trace_mutex_release(void) {
    trace_mutex.unlock();
}

// debug printf function
extern "C"  unsigned int rt_time_get (void);

extern "C"  void trace_printer(const char* str) {
    unsigned int clk = rt_time_get();
    printf("%8u %s\r\n", clk, str);
}

extern "C"  char* trace_prefix(size_t sz) {
    return (char*)" ** ";
}

static void app_trace_setup(void) {
    mbed_trace_init();
    mbed_trace_print_function_set(trace_printer);
    mbed_trace_prefix_function_set(trace_prefix);
    mbed_trace_mutex_wait_function_set(trace_mutex_wait);
    mbed_trace_mutex_release_function_set(trace_mutex_release);
    // TRACE_ACTIVE_LEVEL_INFO , TRACE_ACTIVE_LEVEL_ALL
    // TRACE_MODE_COLOR  or TRACE_MODE_PLAIN
    // TRACE_CARRIAGE_RETURN
    mbed_trace_config_set(DBG_DFT_MBED_LOG_LEVEL|TRACE_MODE_COLOR);
}
```

5.3. IoT Soft Box Library Configuration

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

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

Tunable parameters are:

- LOC_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)