HOP Ubiquitous Documentation Documentation

Release 1.4.0

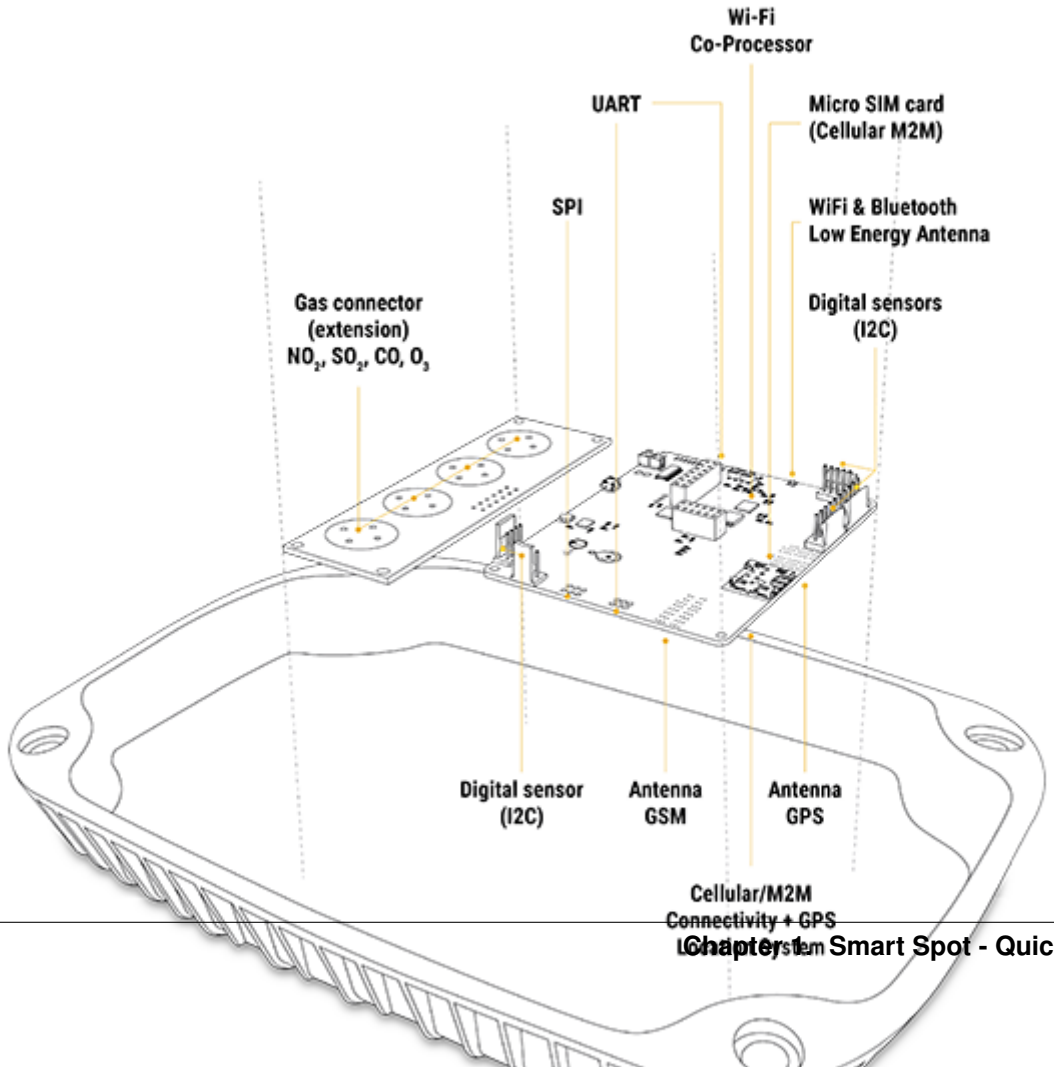
HOP Ubiquitous

Apr 04, 2018

Contents

CHAPTER 1

Smart Spot - Quick Guide



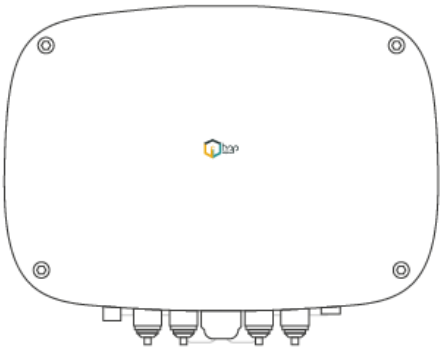
1.1 What is a Smart Spot?

Smart Spot is a connected device that has the ability to create an area of interaction with citizens and visitors. These devices send a URL (website link to open it in any browser such as smartphone Internet browser) to allow users to connect to online contents and discover websites from physical locations. In addition, Smart Spot can optionally capture data regarding energy consumption, noise level and crowd-monitoring (people counting based on WiFi detection); thereby, Smart Spot allows to monitor, define metrics and estimate the comfort level in a city area.

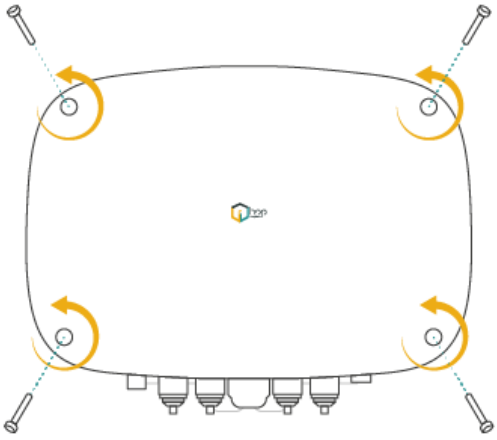
Smart Spot contextualizes the information of each Point of Interest sending notifications by proximity to mobile phones through Bluetooth Low Energy in a range of 1 to 80 meters, without requiring to installing any native App over Android OS or iOS, since it is directly supported by the Operating Systems via Physical Web and iBeacon technologies.

1.2 Smart Spot Features

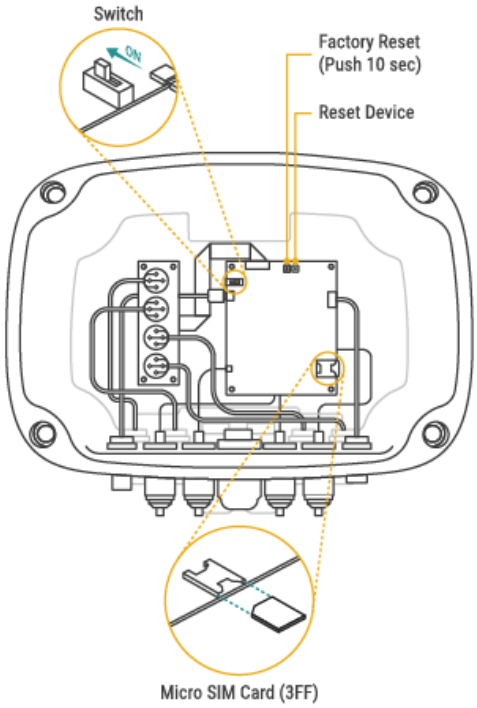
1.3 First steps with your Smart Spot (Set up connectivity)



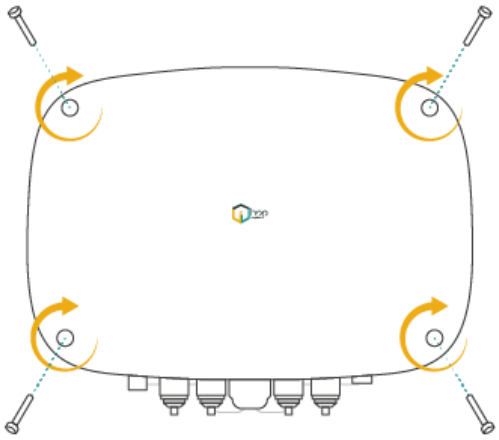
1. This is a Smart Spot.



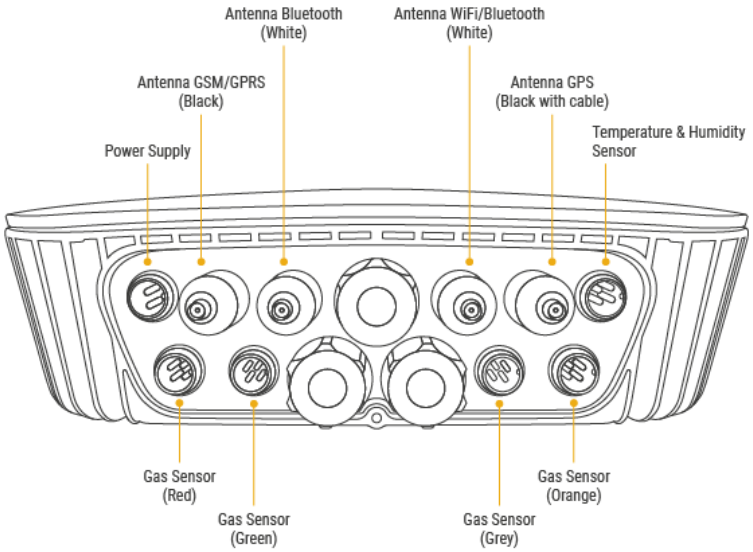
2. On top of the Smart Spot there are four screws. Unscrew the cover and open the Smart Spot.



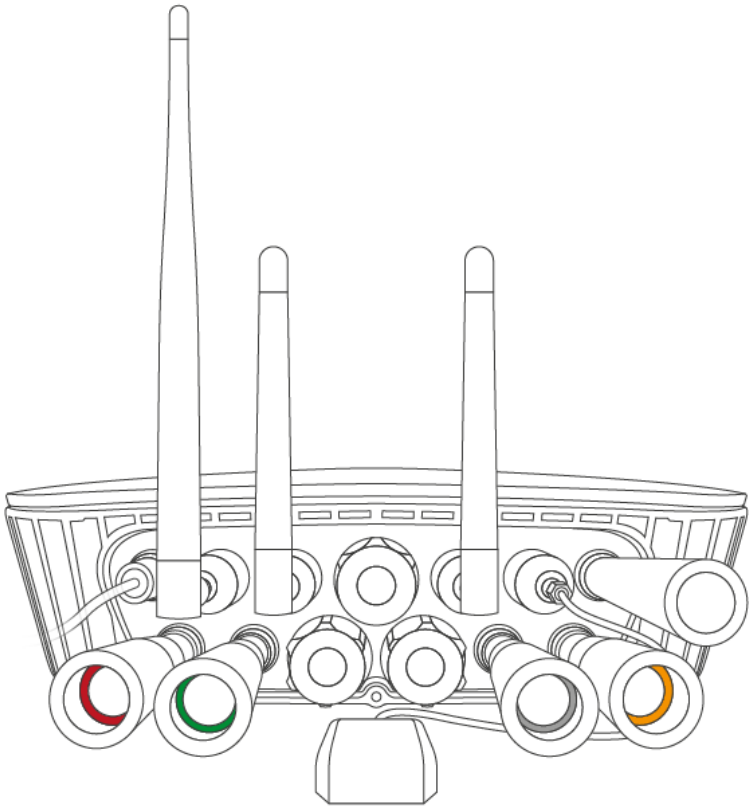
3. When the Smart Spot is open, make sure the switch is on and insert the Micro SIM Card in its place.



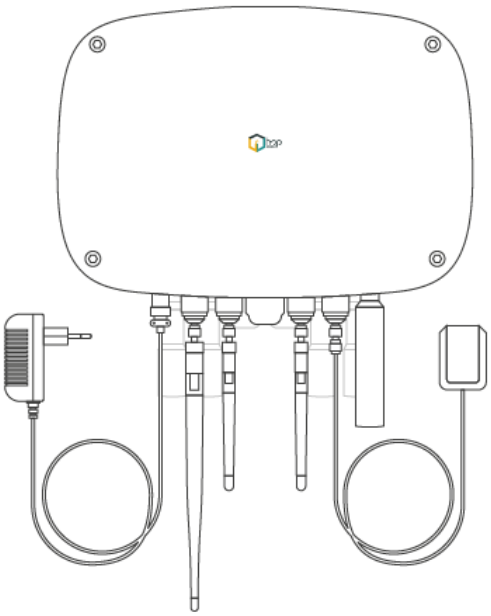
4. Put the cover back and place the screws in their place.



5. When everything is ready inside the Smart Spot, you can start putting the probes and antennas on the back of the device following the scheme of this figure.

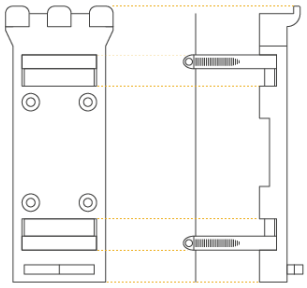


6. Once all the probes and antennas are placed, the appearance of the Smart Spot would look like the figure.

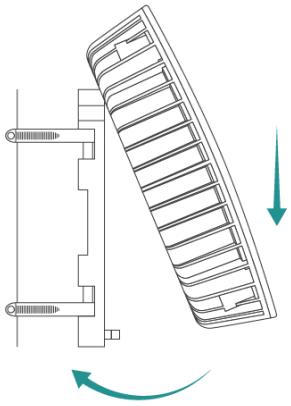


7. Finally, this is the aspect of a Smart Spot with all peripherals properly placed.

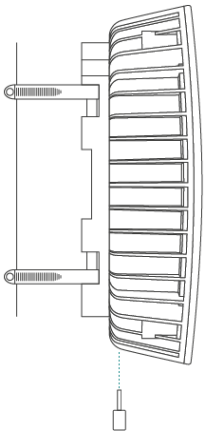
1.4 How to install the device on a pole or on a wall?



1. Put the clamps around the pole to hold the support. Screw the clamps until the support is fixed. On wall, hold the support on the wall directly.



2. Put the device on the support fitting the three pins in the three slots that the Smart Spot has in the back. Then, drop it and fit it into the lower pin.



3. Once fitted on all the pins, insert the screw through the bottom and press it until the Smart Spot is fixed.

1.5 What is Homard Dashboard?

homard
OMA Resources Dashboard

HOMARD is a device management platform for the OMA LwM2M protocol. HOMARD is our own platform.

The platform offers functionalities for device management, i.e., remote maintenance, fi rmware upgrade and open/standard APIs for information reporting.

1.5.1 Software Management

Enabling the installation, removal of applications, and retrieval of inventory software components already installed on the device and the most relevant fi rmware upgrades over the air.

1.5.2 Diagnostics and monitoring

Enabling remote diagnostics and data models to check devices status, memory status, battery status, radio performance, Quality of Services (QoS) parameters, peripherals status, sensors status, and other relevant parameters for remote monitoring.

1.5.3 Connectivity and Security

Bluetooth, cellular connectivity, proxies, list of authorized servers for remote fi rmware upgrade and also all the relevant parameters for enabling secure communication.

1.5.4 Device Capabilities

Allowing to the Management Authority to remotely enable and disable device peripherals like cameras, Bluetooth, USB, sensors (ultrasound, temperature, humidity, etc.) and other relevant peripherals from the nodes.

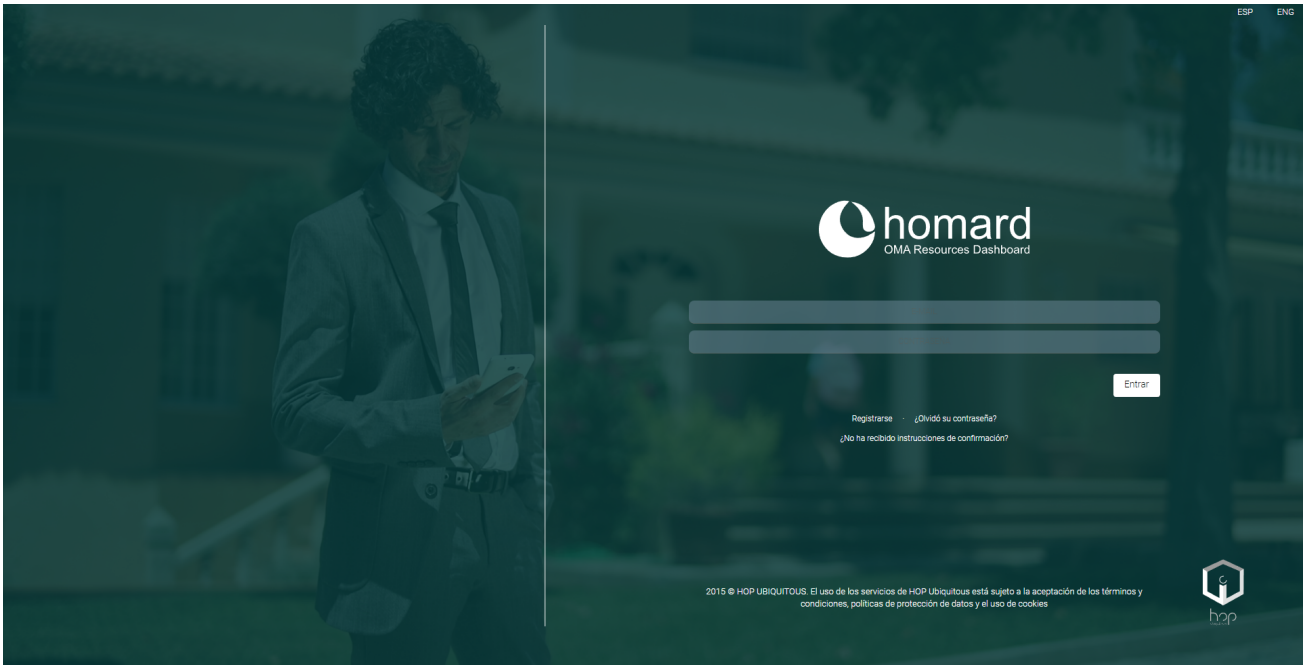
1.5.5 Lock and wipe

Allowing to remotely lock and/or wipe the device, for instance when the device is lost (relevant for devices in open air, outdoor, etc.), or when the devices are stolen or sold. It enables the remote erase of personal/enterprise data when they are compromised.

1.5.6 Policy management

Allowing the deployment on the device policies which the client (node, device, sensor) can execute and enforce independently under some specifi c conditions, i.e., if some events happen, then it performs some operations.

1.6 Connecting and managing the device (default configuration)



The Smart Spots have a pre-configured WiFi Station where devices will connect. This Access Point (AP) can be easily deployed from GSM/GPRS Routers, MiFi Routers, WiFi Access Points, etc.

The default configuration is:

- SSID name: **defaultSSAP**
- Password: **defaultSSAP1234**

It means that Smart Spot will search and try to connect by default to a WiFi network with the mentioned SSID and password.

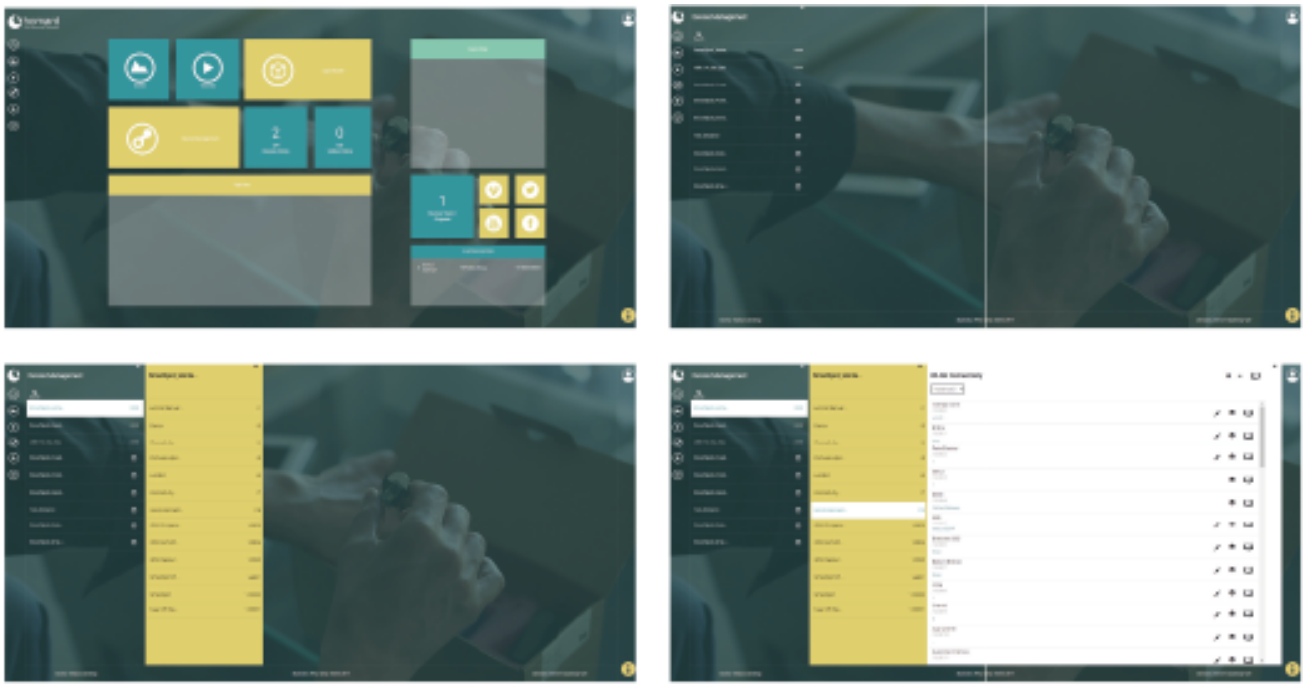
You need to create an access point in your smartphone or router with the default SSID name and Password for the device to connect to the network. When this access point is created, the device will connect automatically and it will be visible in Homard Dashboard.

From http://staging.hopu.eu, enter with your User and Password that we give you and can see the Homard Dashboard options.

There are two ways to connect the Smart Spot to network other than default network:

- WiFi Connectivity.
- APN Connectivity (SIM Card).

1.6.1 WiFi Connectivity



To connect to a network other than default network:

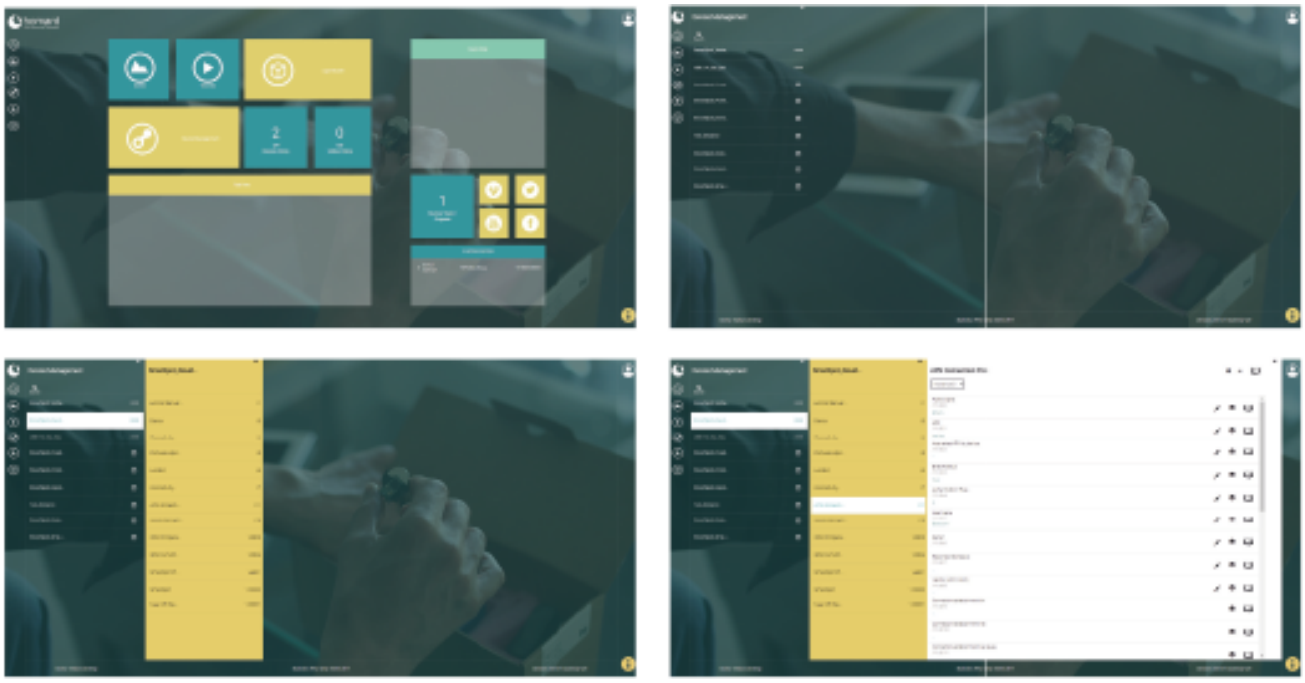
1. Firstly create an access point with the default configuration of device.
2. Go to **Device Management** and you will see a list of devices connect to the network.
3. Locate the device to be configured and click it.
4. In the next column, a list of object of this device appears, therefore search the **WLAN Connectivity** object.
5. Create a new Instance and enter data:

- **Instance ID:** Number not equal to zero.
- **Interface name:** Human-readable identifier.
- **Enable (True or False):** True to connect with this configuration.
- **SSID:** SSID name of network to connect the device.
- **Mode:** Normally must be 1 (client), but there are others options.
- **Channels:** Choose a number of channel.
- **Standard:** Network type. Normally is the option 2.
- **Authentication Mode:** Resource can be anyone. It will be automatically configured by the stack.
- **Encryption Mode:** Defines which key is expected and used: * **WPA2/WPA:** Will require fi ll the "WPA Pre Shared Key" field. * **WEP:** Will require fi ll the "WEP Key Index" and "WEP Key Index" must be 1.

6. In the list of objects, search Device object and click it.
7. Search the Reboot option and click on play icon.

Finally, the device will reboot and connect to the configured network. Note that multiple access points configurations can be stored and Smart Spot will try to connect to all of them until that it gets Internet connectivity.

1.6.2 APN Connectivity (Cellular / M2M)



To connect the device to an APN network other than the default network:

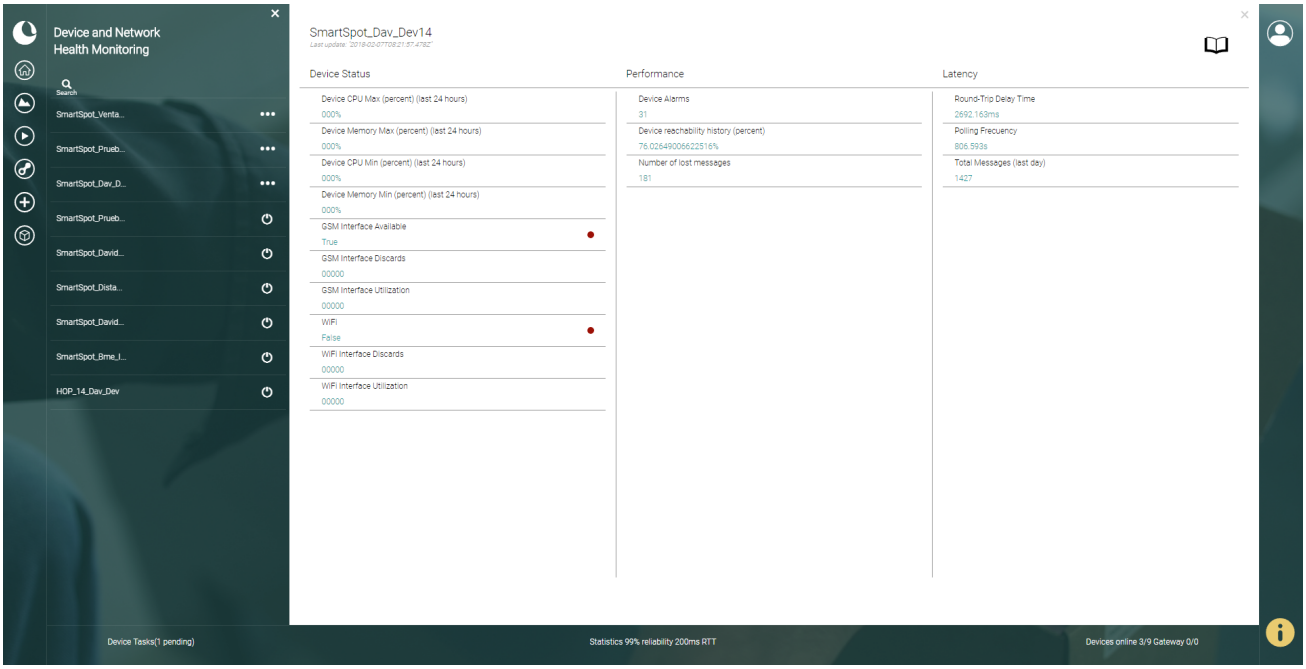
1. Firstly create an access point with the default configuration of device.
2. Go to **Device Management** and you will see a list of devices connect to the network.
3. Locate the device to be configured and click it.
4. In the next column, a list of object of this device appears, therefore search the **APN Connection** object.
5. Create a new Instance and enter data:

- **Profile name:** This name cannot be longer than 16 characters. Identifies the profile name on the device.
- **APN:** This resource contains the APN which will be used to achieve cellular connectivity. This value is provided by the SIM card telco provider.
- **Enable Status:** This resource allows to enable o disable the configuration. Usually it will be true. The OMA Object only allows to add one instance corresponding to the SIM card inserted on the device. Modify or delete/add the instance to change the configuration.
- **Authentication Type:** This resource has a value >=0 and <=3. The value introduced by default should be 0. The authentication type will be automatically selected based on the connectivity provider.
- **User name:** Resource is used to specify the user name used to access the APN. Unless specified by the telco provider, this resource could be empty.
- **Secret:** Resource is used to specify the user name used to access the APN. Unless specified by the telco provider, this resource could be empty.
- **SIM card PIN:** Resource is used to introduce the PIN used to unblock the SIM card.

6. In the list of objects, search Device object and click it.
7. Search the Reboot option and click on play icon.

Finally, the device will reboot and connect to the configured APN.

1.7 How to verify that the sensor is properly working?



To ensure that device and sensors are working, you can enter to **Health Monitoring** section of Homard. Here you will fi nd information about **Device Status**, **Performance** and **Latency**.

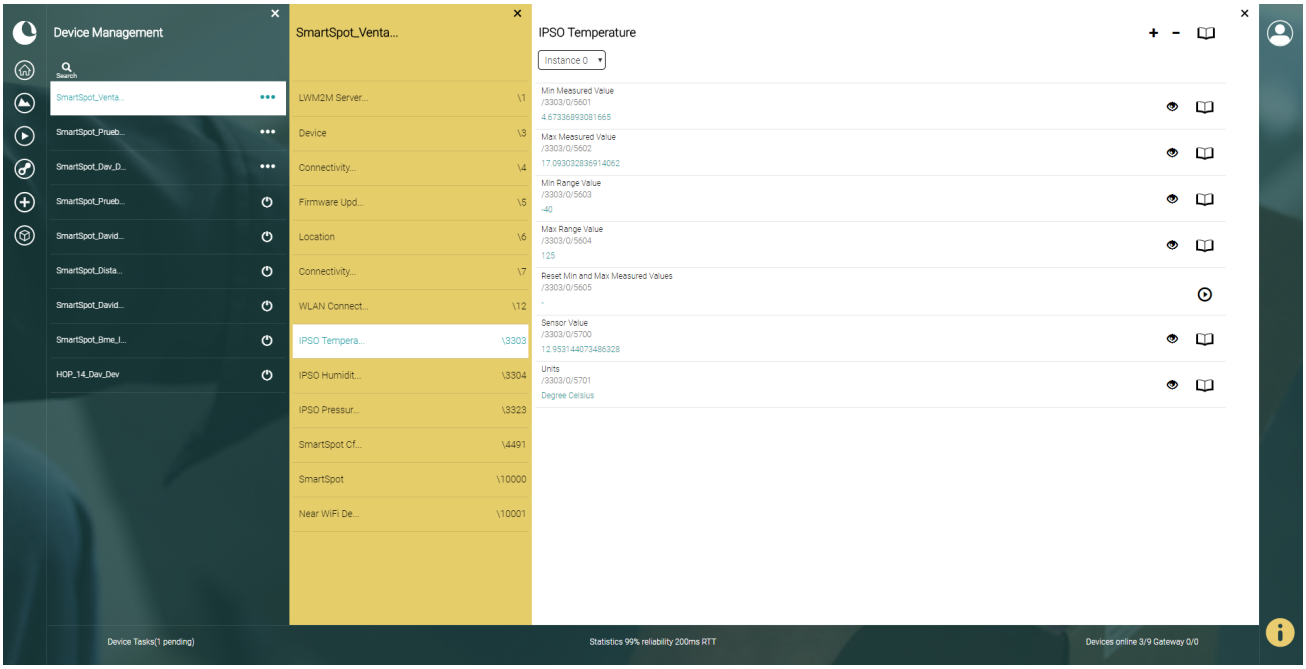
In Device Status you can observe data about the Smart Spot CPU and Memory consumption and the configuration of the device.

The Performance section shows information about the Smart Spot reachability, lost messages and other alarms.

Finally, Latency section shows information about the time it takes to perform a test and communicate with the server.

To update the information you need to click on the Book icon at the top right of the page. The information will be visible at that time.

1.8 How to see the sensor information?



To see the information for each of sensor, you need to ingress to **Device Management** section of Homard and, in the list of objects, look for the sensor that you want information. In this figure you can see the **Temperature Object** and all its resources.

Here, you can see the information for each resource of Temperature Sensor.

To update the information of all resources you need to click on the Book icon at the top right of the page. If you want update the information of a single resource, you need to click on the Book icon at right of the resource name.

Here, you can also do more actions with the resources.

You can create a simple observer or composite observer in a resource clicking the Eye icon at the right of the resource name. This will send notifi cations every time that these resource changes.

Finally, you can active actions such as start/stop or reset. To active this actions you need to click on the Play icon at the right of the resource name.

CHAPTER 2

Smart Spot - Development Guide

2.1 Homard API REST

Homard offers in top of the OMA LwM2M server a RESTFul services such as mechanism to communicate with the OMA LwM2M server. This HTTP/HTTPS API RESTFul allows users to manage connected devices connected to the server.

Homard shows connected devices, and manages each of them. Inherit from OMA, the devices expose a set of objects/resources, which can contain one or more instances, and each instance, contains the final resources. Some objects allow multiple instances (such as a Digital I/O Object, which represents the different digital inputs and outputs from the Smart Spot, which can be used to control relays, read external added digital sensors etc.) and others only one instance (such as Device Object). Numbers are used to identify the tuple (Object ID / Instance ID / Resource ID).

There are two API types: Synchronous and Asynchronous:

- **Synchronous API** provides the data as part of the reply to a request in near-real time; it is used mainly for data which is available in Homard platform internally. Therefore, non-external delays are introduced and a very fast reply can be offered.
- **Asynchronous API** is required mainly for the data coming from the sensors (i.e. Smart Spot); since the value will be available when the Smart Spot replies to the request to Homard; delays can be introduced due to issues such as communication latency (GPRS, WiFi etc. latency), sensors reading time (specially sensors such as air quality which includes a processing time of few seconds), and finally due to duty cycle from the Smart Spot in order to optimize energy (battery or solar panels powered). For that reason, Asynchronous API will requires a callback address to inform when the value is available.

Information returned by the API is encapsulated into a JSON data structure, since is the best way to communicate with high level applications (commonly developed in Java, JavaScript and Android).

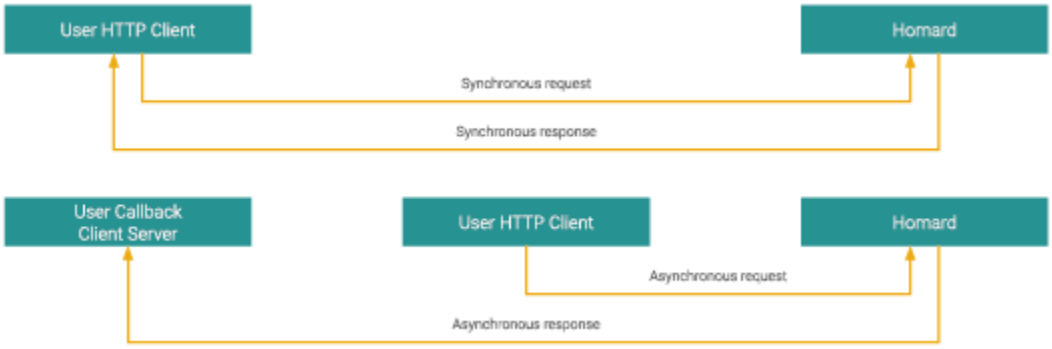
More info in: https://homard.hopu.eu/indexPage/wiki.html

HTTP PATH	Method	Description	Result
/api/rest	GET	Returns all OMA clients connected to HOMARD	[{client1},{client2},...{clientN}
/api/rest/{endpoint_id}	GET	Returns the client description for a {endpoint_id}	{client}
/api/rest/{endpoint_id}/{resource}?callback=http{s}&resource={DESTINATION_URL}	GET	Returns {resource} from client {endpoint_id} to destination callback URL	[{resource1},{resource2},...,{resour
/api/rest/{endpoint_id}/{resource}?callback=http{s}&resource={DESTINATION_URL}	PUT	Changes the value of the resource {resource} of client {endpoint_id}. Content type need to be application/JSON and in the request body {"id": 2, "value": new_value}	
/api/rest/{endpoint_id}/{resource}?cbmin=&cbmax=&callback=http{s} (Requires POST JSON payload)	POST	Sets observe on the resource {resource} and, when the resource value is between 'cbmin' and 'cbmax', posts its value on 'URL'. Note that if cbmin and cbmax are omitted then all values will be tracked.	
/api/rest/{endpoint_id}/{resource}?callback=http{s}	DELETE	Stops the monitoring on resource {resource}	

2.1.1 API-RESTFul

There are two API types synchronous and asynchronous, on one hand it is synchronous API, that corresponds with Server request such as get the client list, get a client specific data or list the observes set on a device. On the other hand it is device request. The device requests are launched on background, this means that the response will send to a callback URL defined by the client. This API requires the HTTP basic authentication using a Homard account.

The following diagram shows the differences between synchronous and asynchronous API.



This subsection will present the synchronous API and an example. The available synchronous API resources are as follows:

HTTP METHOD	Path	Description
GET	/api/rest/	Lists all connected clients with their current data
GET	/api/rest/{endpoint_id}	Get specific connected client data
GET	/api/rest/cached/{endpoint_id}/{resource}	Read the resource cached value (last value)
GET	/api/rest/{endpoint_id}/co	List client observes
POST	/api/rest/{endpoint_id}/{resource}/observe	Observes the selected resource and updates the internal cache.
DELETE	/api/rest/{endpoint_id}/{resource_path}/observe	Removes selected client observer
DELETE	/api/rest/{endpoint_id}/{resource_path}/co/{coID}	Removes selected client composite observer

2.1.2 Cached Synchronous RESTFul API

Synchronous APIs are for requests which are directly resolved by Homard Server. Thereby, we get the last values cached on the server directly, without any delay. At this way, Homard can be seen as a data broker.

HTTP METHOD	Path	Parameters	Description
GET	/api/rest /cached /{endpoint_id}/{resource}	None	Read the resource cached value

The API is very useful in special for applications that can not setup a HTTP server in its architecture. This synchronous API can be outdated, to solve it, the user can enable a simple observer, which constantly updates the resource. In order to create an internal observer to maintain the value updated for a specific resource is as follows:

HTTP METHOD	Path	Parameters	Description
POST	/api/rest /{endpoint_id}/{resource}/observe	None	Observes the selected resource and updates the internal cache

2.1.3 Asynchronous RESTFul API

This subsection present the asynchronous API and an execution example. The available asynchronous API resources are as follows:

HTTP METHOD	Path	Parameters	Description
GET	/api/rest/{endpoint_id}/{resource}	**cburl:** Destination callback url	Read the value and returns it to the destination callback url
		proto: Destination callback url protocol	
PUT	/api/rest/{endpoint_id}/{resource}	**cburl:** Destination callback url	Write the value on resource and returns the status to callback url
		proto: Destination callback url protocol	

2.1.4 Historical Data RESTFul API

Homard offers in top of the OMA LwM2M server a RESTFul services such as mechanism to communicate with the OMA LwM2M server. This HTTP/HTTPS RESTFul API allow users to query the stored information such as temperature and humidity. In addition, sensor status and maintenance parameters can be queried in order to know if there are any problem with network connectivity.

HTTP PATH	Method	Description	Result
/api/hist	GET	Returns historic of OMA clients connected to HOMARD.	[{ “name”: “HOPf4b85eab98de”, “cn”: “HopCore2” }, { “name”: “HOPf4b85eab98d6”, “cn”: “HopCore9” }]
/api/hist/{endpoint_id}[?from={timestamp}][&to={timestamp}][&limit={max_records}][&page={limited_page_requested}]	GET	Returns historic values of {enpoint_id} device.	[{ “name”: “HOPf4b85eab98de”, “temperature”: 27.36767578125, “humidity”: 45.654815673828125, “date”: “Jun 21, 2016 2:10:36 PM” }, . . .]
/api/hist/{endpoint_id}/temp[?from={timestamp}][&to={timestamp}][&limit={max_records}][&page={limited_page_requested}]	GET	Returns historic values of {enpoint_id} device.	{ values:[27.36767578125, 27.96767578125 . . .], dates:[“Jun 21, 2016 2:10:36 PM”, “Jun 21, 2016 2:10:37 PM”, “Jun 21, 2016 2:10:48 PM”, ...] }
/api/hist/{endpoint_id}/hum[?from={timestamp}][&to={timestamp}][&limit={max_records}][&page={limited_page_requested}]	GET	Returns historic values of {enpoint_id} device.	{ values:[27.36767578125, 27.96767578125 . . .], dates:[“Jun 21, 2016 2:10:36 PM”, “Jun 21, 2016 2:10:37 PM”, “Jun 21, 2016 2:10:48 PM”, ...] }
/api/hist/events/{endpoint_id}/[?from={timestamp}][&to={timestamp}][&limit={max_records}][&page={limited_page_requested}]	GET	Returns events such as when {enpoint_id} was registered, updated or deregistered.	[{ “name”: “HOPf4b85eab98de”, “event”: “REGISTRATION”, “ts”: “Jun 21, 2016 2:08:09 PM” }, { “name”: “HOPf4b85eab98de”, “event”: “UPDATED”, “ts”: “Jun 21, 2016 2:08:56 PM” }, . . .]
/api/hist/events/recount/{endpoint_id}/[?from={timestamp}][&to={timestamp}]	GET	Returns {enpoint_id} events.	[{ “count”: 2, “event”: “DEREGISTRATION” }, { “count”: 4, “event”: “REGISTRATION” }, { “count”: 841, “event”: “UPDATED” }]

2.2 Homard RESTFul API Examples

2.2.1 Synchronous RESTFul API Examples

List connected clients: GET /api/rest/

Query:

```
GET /api/rest
```

Result:

```
[
    {
        “endpoint”:”HOP-Sensor-Debug”,
        “registrationId”:”MTKt5ejSeu”,
        “registrationDate”:”2015-02-16T01:06:27+01:00”,
        “address”:”/127.0.0.1:54604”,
        “lwM2MmVersion”:”1.0”,
        “lifetime”:120,
        “bindingMode”:”UQ”,
        “rootPath”:”/”,
        “objectLinks”:[
            {
                “url”:”/1/0”,
                “attributes”:{
                },
                “objectId”:1,
```

```
                "objectInstanceId":0
            },
            {
                "url":"/3",
                "attributes":{
                },
                "objectId":3
            },
            {
                "url":"/4",
                "attributes":{
                },
                "objectId":4
            },
            {
                "url":"/5",
                "attributes":{
                },
                "objectId":5
            },
            {
                "url":"/3201/0",
                "attributes":{
                },
                "objectId":3201,
                "objectInstanceId":0
            },
            {
                "url":"/3201/1",
                "attributes":{
                },
                "objectId":3201,
                "objectInstanceId":1
            }
        ]
    }
]
```

Get specific client: GET /api/rest/{endpoint_id}

Query:

```
GET /api/rest/HOP-Sensor-Debug
```

Result:

```
{
    "endpoint":"HOP-Sensor-Debug",
    "registrationId":"MTKt5ejSeu",
    "registrationDate":"2015-02-16T01:06:27+01:00",
    "address":"/127.0.0.1:54604",
    "lwM2MmVersion":"1.0",
    "lifetime":120,
    "bindingMode":"UQ",
    "rootPath":"/",
    "objectLinks":[
        {
            "url":"/1/0",
```

```
            "attributes":{
            },
            "objectId":1,
            "objectInstanceId":0
        },
        {
            "url":"/3",
            "attributes":{
            },
            "objectId":3
        },
        {
            "url":"/4",
            "attributes":{
            },
            "objectId":4
        },
        {
            "url":"/5",
            "attributes":{
            },
            "objectId":5
        },
        {
            "url":"/3201/0",
            "attributes":{
            },
            "objectId":3201,
            "objectInstanceId":0
        },
        {
            "url":"/3201/1",
            "attributes":{
            },
            "objectId":3201,
            "objectInstanceId":1
        }
    ]
}
```

Get client observes: GET /api/rest/{endpoint_id}/co

Query:

```
GET /api/rest/UBI8086f2759cbb/observes
```

Result:

```
[
    {
        "client": "HOPf4b85eab962b",
        "path":
            {
                "objectId": 3303,
                "objectInstanceId": 0,
                "resourceId": 5700
            },
        "ocos":
```

```
            [
                {
                    "url": "homard.hopu.eu:8090/co",
                    "protocol": "https://",
                    "threshold": 0,
                    "condition": ">=",
                    "method": "POST",
                    "oneShot": false,
                    "outputFormat":
                        [
                            {
                                "field": "sensor",
                                "value": "$eID"
                            },
                            {
                                "field": "value",
                                "value": "$value"
                            },
                            {
                                "field": "r",
                                "value": "$resource"
                            },
                            {
                                "field": "threshold",
                                "value": "$threshold"
                            }
                        ],
                    "id": 1,
                    "type": 0,
                    "endpoint": "HOPf4b85eab962b",
                    "path": "/3303/0/5700"
                }
            ]
        }
    }
]
```

Remove client observe: DELETE /api/rest/{endpoint_id}/{resource_path}/observe

Query:

```
DELETE /api/rest/HOPf4b85eab9b03/3303/0/5700/observe
```

Result:

```
HTTP Response 200 OK
```

2.2.2 Asynchronous RESTFul API Examples

Read device resource: GET /api/rest/{endpoint_id}/{resource}

GET parameters:

- **proto** (mandatory): Determines the protocol to use (HTTP or HTTPS).
- **cburl** (mandatory): Determines the response destination url.
- **cbauthusr** (optional): Used if the destination server requires Basic authentication.

- **cbauthpass** (optional): Used if the destination server requires Basic authentication.

Read Device Resource Query:

```
GET /api/rest/HOPf4b85eab9b03/1/0/5?proto=https&cburl=homard.hopu.eu:8090/co
```

Post Result on Callback Url:

```
{
    "eid":"HOPf4b85eab98de",
    "url":"/1/0/5",
    "oID":1,
    "iID":0,
    "rID":5,
    "type":"INTEGER",
    "value":1
}
```

Post Error on Callback Url:

```
{
    "eid":"HOPf4b85eab98de",
    "operation":"read",
    "resource":"/1/0/5",
    "status":"ERROR"
}
```

Write device resource: PUT /api/rest/{endpoint_id}/{resource}

PUT parameters: proto (mandatory), cburl (mandatory), cbauthusr (optative), cbauthpass (optative).

Write Device Resource Query:

```
PUT /api/rest/HOPf4b85eab9b03/1/0/5?proto=https&cburl=homard.hopu.eu:8090/co
```

Required Payload (Data to write):

```
{
    "id": 5,
    "value": 2
}
```

Post Result on Callback Url:

```
{
    "eid":"HOPf4b85eab98de",
    "url":"/1/0/5",
    "oID":1,
    "iID":0,
    "rID":5,
    "type":"INTEGER",
    "value":1
}
```

Post Error on Callback Url:

```
{
    "eid":"HOPf4b85eab98de",
    "operation":"write",
```

```
    "resource":"/1/0/5",
    "status":"ERROR"
}
```

2.2.3 Asynchronous API: Observes

Observation creation requires a specific POST payload. This is a JSON object that contains the necessary parameters. The necessary data to create an observer is as follows:

- Observe resource: POST /api/rest/{endpoint_id/{resource}/co
- **Payload:**
  - **threshold:** Threshold value
  - **op:** Operation condition (<=, >=, . . .)
  - **typeShot:** Type shot could be "repeat" or "oneshot"
  - **method:** The method of the request, usually POST
  - **durl:** Destination URL where the observer will send the notifications
  - **proto:** Protocol to use ("https://" or "http://")
  - **authUsr:** Destination URL Basic authentication user (Optional)
  - **authPass:** Destination URL Basic authentication password (Optional)

Also the JSON schema for this object is:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "http://jsonschema.net",
    "type": "object",
    "properties": {
        "threshold": {
            "id": "http://jsonschema.net/threshold",
            "type": "string"
        },
        "op": {
            "id": "http://jsonschema.net/op",
            "type": "string"
        },
        "typeShot": {
            "id": "http://jsonschema.net/typeShot",
            "type": "string"
        },
        "method": {
            "id": "http://jsonschema.net/method",
            "type": "string"
        },
        "durl": {
            "id": "http://jsonschema.net/durl",
            "type": "string"
        },
        "proto": {
            "id": "http://jsonschema.net/proto",
            "type": "string"
        },
```

```
        "authUsr": {
            "id": "http://jsonschema.net/authUsr",
            "type": "string"
        },
        "authPass": {
            "id": "http://jsonschema.net/authPass",
            "type": "string"
        }
    },
    "required": [
        "threshold",
        "op",
        "typeShot",
        "method",
        "durl",
        "proto"
    ]
}
```

Example of request

Create observation:

```
POST /api/rest/HOPf4b85eab9b3a/3303/0/5700/co
```

Required Payload:

```
{
    "threshold": "0",
    "op": ">=",
    "typeShot":"repeat",
    "method": "POST",
    "durl":"homard.hopu.eu:8090/co",
    "proto":"https://",
    "authUsr": "",
    "authPass": ""
}
```

Response Received:

```
HTTP response 200
```

Notification Received on Callback Url:

```
{
    "eid":"HOPf4b85eab9b3a",
    "url":"3303/0/5700",
    "oID":3303,
    "iID":0,
    "rID":5700,
    "value":"17.028683"
}
```

Error Notification Received on Callback Url:

```
{
    "eid":"HOPf4b85eab9b3a",
    "operation":"observe",
```

```
    "resource":"/1/0/5",
    "status":"ERROR"
}
```

2.2.4 Asynchronous API: Customizing observer notification message (integrating with third party platform that requires a specific format)

Observation notifications have a standard output that can be formatted according to user needs. To achieve this we must add a JSON array called "outputPacket" that contains the format of the notification packet.

This array contains field/value JSON objects This array contains one or more objects of type field/value ({"field":"NameFile", "value":"Variable or Constant"}). The field "field" contains the attribute name of the notification final object. The field "value" represents the value of that attribute, It can be constant or variable. The allowed variable are the following:

- **$eID:** The Endpoint ID of the observed device.
- **$resource:** The OMA LwM2M resource observed.
- **$value:** The observation value received.
- **$threshold:** The observation threshold.
- **$condition:* The observation condition.

For example if we want to receive the following notification:

```
{
    "sensor": "HOPf4b85eab9b3a",
    "value":15.32,
    "uuid": "067e6162-3b6f-4ae2-a171-2470b63dff00"
}
```

We have to set the following "outputPacket":

```
{
    ...,
    "outputPacket":[
        {
            "field":"name",
            "value":"$eID"
        },
        {
            "field": "value",
            "value":"$value"
        },
        {
            "field": "uuid",
            "value":"067e6162-3b6f-4ae2-a171-2470b63dff00"
        }
    ]
}
```

The JSON schema of the complete object is:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "http://jsonschema.net",
```

```
    "type": "object",
    "properties": {
        "threshold": {
            "id": "http://jsonschema.net/threshold",
            "type": "string"
        },
        "op": {
            "id": "http://jsonschema.net/op",
            "type": "string"
        },
        "typeShot": {
            "id": "http://jsonschema.net/typeShot",
            "type": "string"
        },
        "method": {
            "id": "http://jsonschema.net/method",
            "type": "string"
        },
        "durl": {
            "id": "http://jsonschema.net/durl",
            "type": "string"
        },
        "proto": {
            "id": "http://jsonschema.net/proto",
            "type": "string"
        },
        "authUsr": {
            "id": "http://jsonschema.net/authUsr",
            "type": "string"
        },
        "authPass": {
            "id": "http://jsonschema.net/authPass",
            "type": "string"
        },
        "outputPacket": {
            "id": "http://jsonschema.net/outputPacket",
            "type": "array",
            "items": [
                {
                    "id": "http://jsonschema.net/outputPacket/0",
                    "type": "object",
                    "properties": {
                        "field": {
                            "id": "http://jsonschema.net/outputPacket/0/field",
                            "type": "string"
                        },
                        "value": {
                            "id": "http://jsonschema.net/outputPacket/0/value",
                            "type": "string"
                        }
                    }
                },
                {
                    "id": "http://jsonschema.net/outputPacket/1",
                    "type": "object",
                    "properties": {
                        "field": {
                            "id": "http://jsonschema.net/outputPacket/1/field",
```

```
                            "type": "string"
                        },
                        "value": {
                            "id": "http://jsonschema.net/outputPacket/1/value",
                            "type": "string"
                        }
                    }
                }
            ]
        }
    },
    "required": [
        "threshold",
        "op",
        "typeShot",
        "method",
        "durl",
        "proto"
    ]
}
```

Example of request

Create Observation:

```
POST /api/rest/HOPf4b85eab9b3a/3303/0/5700/observe
```

Required Payload:

```
{
    "threshold": "0",
    "op": ">=",
    "typeShot":"repeat",
    "method": "POST",
    "durl":"homard.hopu.eu:8090/co",
    "proto":"https://",
    "authUsr": "",
    "authPass": "",
    "outputPacket":[
        {
            "field":"name",
            "value":"$eID"
        },
        {
            "field": "value",
            "value":"$value"
        }
    ]
}
```

Response Received:

```
HTTP response 200
```

Notification Received on Callback Url:

```
{
    "name":"HOPf4b85eab9b03",
```

```
    "value":17.028683
}
```

2.3 URL Manager (Physical Web configuration for advertising URLs)

Device URL Manager is the key component of the Industrial Physical Web solution offered to provide accessible and intuitive user interfaces. In details, the Device URL manager is used to administer the URL broadcasted / transmitted by the devices; this URL can be issued by BLE or Wi-Fi direct and must be coded with the Eddystone URL protocol1 from Google.

Nowadays, there is more than 3 million Apps hosted on Google Play, most of them are used a few times for their temporal or location context and later they are forgotten wasting resources on mobile devices or at best cases uninstalled. Google wants to solve this problem through Physical Web, this technology will allows service providers to interact with users depending on the location, temporality context or directly with the objects surrounding the user without the need of install any application on any device, These applications will be developed as progressive webs and they will allow user to feel that they are interacting with the real world through native applications, these applications are capable of interacting directly with mobile device hardware or even receiving notifications.

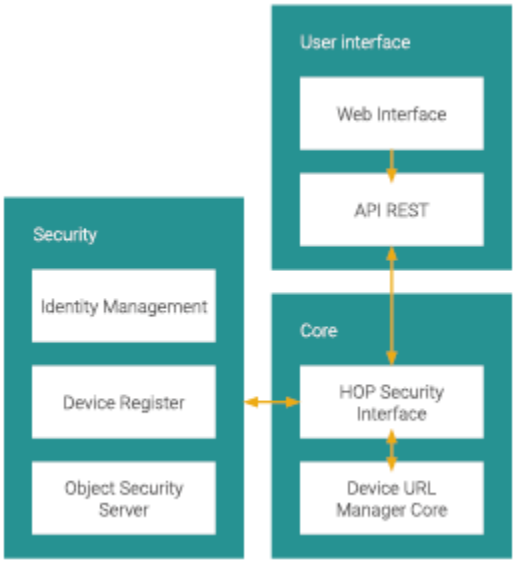
HOP Ubiquitous is a partner from Google for physical Web, and a service for the creation of secure and validated URLs is available in HPOI.info. Contact HOP Ubiquitous team for more details.

Physical Web creates a communication channel that connects the physical and virtual worlds using the Bluetooth Low Energy connection to send "push" notifications to nearby Smartphones that are in their range of action. This connection does not need any tracker native app, only with Google Chrome installed and with bluetooth switched on every user can interact with digital content directly in this physical point.

Thanks to technologies such as physical web and the Device URL Manager, Smart Spot is able to broadcast a URL with temporal and spatial context that will send to people around the Smart Spot for use cases such as tourism, infortainment, accesibility, marketing, make visible content or Webs available linked to a Physical place, etc

Device URL Manager provides an Open Source Core2 for users to be able to develop their own solutions which is accessible through an API REST, which is developed in python using Open Source frameworks such as Django and Django REST Framework that provides to the developers an ecosystem that allows an easy extension of the system for introducing layers of security or registration of devices.

URL Manager: https://github.com/HOP-Ubiquitous/DeviceUrlManager



2.3.1 URL Manager RESTFul API

This tool is used for manage the physical web URL of any device by software.

Smart PhoneS detect the Eddystone URL advertisement with a fixed device url that point to the Device URL Manager, then the Device URL Manager will redirect the request to the real uRL.

More information in the repository: https://github.com/HOP-Ubiquitous/DeviceUrlManager

2.3.2 URL Manager RESTFul API Examples

- **Fixed device url** (Request): https://hpoi.info/AA00BB11DD22
- **Device URL Manager external device URL** (Response): https://google.es
- **MAC address**: is a normal MAC without the two dots. E.g: 00:11:22:33:44:55 > shortened MAC: 001122334455

Create Device

URL	Method	URL Params	Data Params	Description
/api/v1/devices	POST	None	**Type**: application/json	Method to add a device with its MAC and a target URL

Success Response:

```
Code: 201 (Created)
```

```
Content:
{
    "mac":"001122334455",
    "external_url":"https://google.es/"
}
```

Error Response:

```
Code: 400 (Bad Request)
```

```
Content:
{
    "bad_field_name":[error causes]
}
```

Sample Call:

```
{
    url: "/api/v1/devices",
    dataType: "json",
    data: {
        "mac": "001122334455",
        "external_url": "https://google.es/"
    },
    type : "POST",
    success : function(r) {
        console.log(r);
```

```
    }
}
```

Show Device Data

URL	Method	URL Params	Data Params	Description
/api/v1/devices/:shortened_mac	GET	shortened_mac=[String]	None	Method to show the device information

Success Response:

```
Code: 200 (Ok)
```

```
Content:
{
    "mac": "001122334455",
    "external_url": "https://google.es/"
}
```

Error Response:

```
Code: 404 (Not Found)
```

```
Content:
{
    "detail": "Device Not Found",
}
```

Sample Call:

```
{
    url: "/api/v1/devices/001122334455",
    dataType: "json",
    type : "GET",
    success : function(r) {
        console.log(r);
    }
}
```

Update Device Data

URL	Method	URL Params	Data Params	Description
/api/v1/devices/:shortened_mac	PUT	None	**Type**: application/json	Method to update the device information

Success Response:

```
Code: 200 (Ok)
```

Content:

```
{
    "mac": "001122334455",
    "external_url": "https://google.es/"
}
```

Error Response:

```
Code: 404 (Not Found)
```

Content:

```
{
    "detail": "Device Not Found",
}
```

Sample Call:

```
{
    url: "/api/v1/devices/001122334455",
    dataType: "json",
    data: {
        "external_url": "https://google.es/"
    },
    type : "PUT",
    success : function(r) {
        console.log(r);
    }
}
```

2.4 Fiware Integration

FIWARE (www.fiware.org) in an open platform promoted by the European Commision and maintained by the FIWARE Foundation, where HOP Ubiquitous is Gold Member.

FIWARE offers an Open Ecosystem that join different technology enablers for scalable data mangement and make feasible to integrate different services and Internet of Things devices into a common and interoperable framework based on Open Standards. In particular, FIWARE is based on Open Standards such as OMA NGSI for the Services Interface and ETSI ISG CIM for the data models. HOP Ubiquitous is an active member and contributor in ETSI ISG CIM and also an active contributor in OMA; being one of pioneer and main companies working around OMA LwM2M protocol.

In details, FIWARE has a strong role in the Smart Cities market, since the is key to the growth and functionality of Smart Cities, for this reason we are committed to initiatives such as Open and Agile Smart Cities (OASC) association with over 100 cities enrolled and FIWARE technology as the basis for making it feasible.

Smart Spot is a FIWARE-ready device, it means that Smart Spot has been validated, passed a set of tests, participated in plugfests and the most important is supporting the APIs, FIWARE data models based on ETSI ISG CIM and it is fully interoperable and integrated with key components from FIWARE such as Orion Context Broker.

In details, Orion Context Broker is the core of FIWARE platform; since it enables the common integration of heterogenous data sources into a common component, which enables the capacity to carry out advanced queries, cross data among heterogenous domains (e.g., noise with crowd, weather and mobility, etc.), and finally it can exports data to several data analytics components such as Hadoop / COSMOS (Big Data), SHT (Time Series), CKAN (Open Data), MongoDB (Non-structured data), etc.

FIWARE and the solutions from HOP Ubiquitous are contributing to the creation of adapted and standardized solutions to satisfy the described process from the cocreation and citizens engagement to the deployment of solutions based on IoT to reach the digitalization and enhancement of different areas in the city.

Thanks to the LwM2M Bootstrap Server deployed and integrated in the Homard platform, it is really easy to setup the server configuration for a the device. In this way, anyone can deploy its own LwM2M IOTAgent with a public server IP and configure the device to integrate it in FIWARE.

A tutorial about Orion Context Broker has been developed by FIWARE and HOP Ubiquitous, which can be downloaded in: http://goo.gl/o1KXcT

2.4.1 OMA LwM2M IoT Agent

OMA LwM2M is a device management protocol created by the Open Mobile alliance (OMA) which allows the remote manipulation of Internet of Things constrained devices. This complete protocol defines the procedures for provisioning, commissioning and management of a device through the definition of the resources exposed by the device.

The functionality of LwM2M protocol is carried out through a set of basic objects such as "Server", "Security", "Device", "Statistics"... but there are also more specific objects defined by the IPSO Alliance which aims to cover the need common definitions for sensors such as temperature, humidity, presence, etc. Or actuators such as power/light/load control, buzzers, etc.

CoAP (Constrained Application Protocol) defines the message header, request/response codes, message options and retransmission mechanisms, this protocol together with UDP is used by LwM2M as a transport mechanism.

There are a large variety of topologies on the IoT but they have three common parts; devices, routers and backends.

In some scenarios, routers are transparent for the enddevices such as cellular technologies, examples are: GSM, SigFox, NarrowBand IoT or Wi-Fi Hallow, since they are already deployed by Telco's. Others, such as Wi-Fi, Bluetooth, 6LoWPAN and Z-Wave are provided individually with the devices. In HOP Ubiquitous our technology is mainly based on GSM and Bluetooth 4.0.

The FIWARE Foundation counts with its own set of services for the IOT. One of this kind of services is called FIWARE IOT Agent, and we can find one that is fully compatible with our device architecture and connection protocols. HOP Ubiquitous in collaboration with ATOS and Telefonica are maintaining the integration of OMA LwM2M protocol with FIWARE via the Orion Context Broker. It is fully Open Source (URL al IoT Agent de HOPU en GitHub), and it counts with a simple deployment over FIWARE and Linux-based platforms.

In addition, HOP Ubiquitous offers containers and Cloudenabled services with the integration of FIWARE and Orion Context Broker (including OMA LwM2M IoT Agent). Contact HOP Ubiquitous support team for more details.

2.4.2 Orion Context Broker

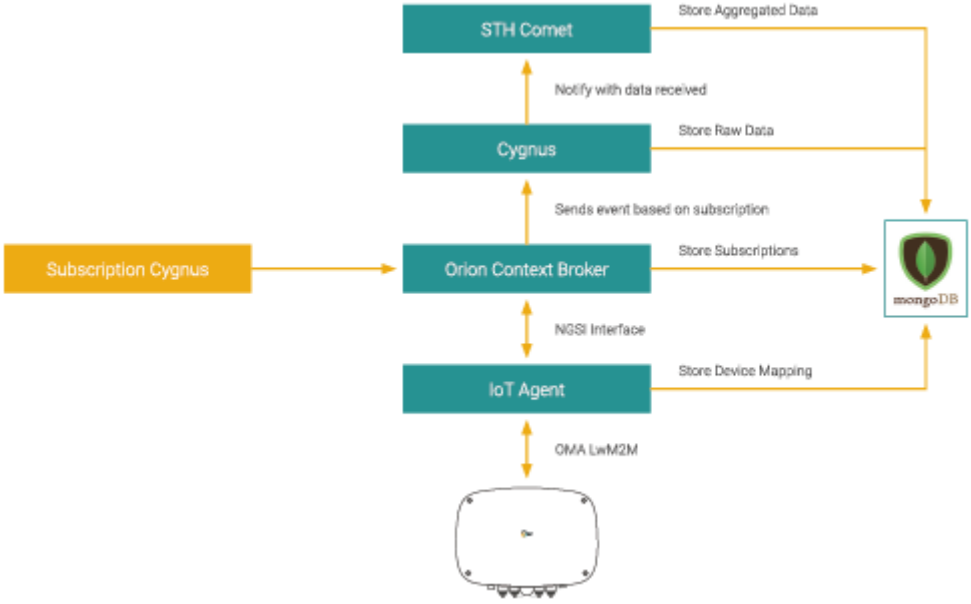
In The architecture, a service for context and information storage, sharing and consumption is needed, in the FIWARE Architecture this is the Orion Context Broker. This service is the one in charge to connect with the OMA LwM2M IOT Agent in order to collect data coming from the IOT devices. One of the most important features of the Context Broker is that it allows to model and gain access to context information in a way that is independent from the source of that information. It uses a non-relational database (Mongodb) to store all the data, and counts with an easy to use REST API with makes data accessible.

2.4.3 Cygnus

This is the FIWARE Service in charge of persisting data. It is based on Apache Flume, Cygnus offers as a data bus the interconnection with several data sources and data platforms such as Hadoop, CKAN, STH Comet etc.

2.4.4 FIWARE architecture

Taking the previous two components into account the FIWARE IoT platform is composed, in the following lines the deployment of every service and the integration with the HOP Ubiquitous Smart Spot is described.



Deployment and integration with FIWARE

A previous setting up is needed before things can be switched on, taking into account the following steps. We will assume that the user has a basic knowledge about DOCKER and LINUX CMD.

- **Get and configure the Hopu modifi ed FiWARE IOTAgent:**
    1. git clone https://github.com/HOP-Ubiquitous/lightweightm2m-iotagent/tree/hopu
    2. change the fi le content for **config.js**, **omaRegistry.json** and **omaInverseRegistry.json** in order to set up our device confi guration to connect with.
    3. execute: npm install
    4. Launch a mongoDB instance for the IOTAgent with external **port 7900**.
- **Prepare The orion context broker infrastructure:**
    1. Launch a mongoDB instance for the IoTAgent with external **port 1026**.
    2. Launch a ORION Context Broker instance.

Docker makes really simple the previous steps.

- **Make sure that the Smart Spot knows where is the IoTAgent to connect with:**
    1. Notify The Smart Spot about the IOTAgent IP with the bootstrap procedure.
- **Rock & Roll:**
    1. Change directory (cd) to the IOTAgent one and execute **./bin/lwm2mAgent.js**.
    2. Turn on the Smart Spot

2.5 Orion Context Broker NGSI RESTFul API

Orion is a C++ implementation of the NGSIv2 REST API binding developed as a part of the FIWARE platform.

Orion Context Broker allows you to manage the entire lifecycle of context information including updates, queries, registrations and subscriptions. It is an NGSIv2 server implementation to manage context information and its availability. Using the Orion Context Broker, you are able to create context elements and manage them through updates and queries. In addition, you can subscribe to context information so when some condition occurs (e.g. the context elements have changed) you receive a notification. These usage scenarios and the Orion Context Broker features are described in this documentation.

2.5.1 Create entity v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to put an entity in a service.

Body:

```
{
    id:”Entity ID”,
    type:”Entity type”,
    “attributeID0”: {
        value:”Attribute value”,
        type:”Attribute type”
    },
    “attributeID1”: {
        value:”Attribute value”,
        type:”Attribute type”
    },...
}
```

Example of Body:

```
{
    id:”Room7”,
    type:”Room”,
    “temperature”: {
        value: 23,
        type: ”Float”
    },
    “preassure”: {
        value: 720,
        type: ”Integer”
    },...
}
```

Return:

```
Return an error message, if already exists an entity in the service with the same id.
```

2.5.2 Retrieve entity v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}	GET	**url**:Link to the service that will be consulted.	Method to retrieve an entity of a service.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity which will be retrieved from the service.	

Return:

```
If the search has been successful then it returns the information else it returns an
↪message error.
```

2.5.3 Retrieve entity as data model v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}?options=keyValues	GET	**url**:Link to the service that will be consulted.	Method to obtain an entity of a service.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity which will be retrieved from the service.	

Return:

```
If the search has been successful then it returns the information of compressed way
↪else returns an error message.
```

2.5.4 Retrieve entities v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities?limit=50	GET	**url**:Link to the service that will be consulted.	Method to retrieve all entities.
		port-orion: Port to connect with the service.	

Return:

```
All entities of the service and for each entity shows their information
```

2.5.5 Retrieve entities as data model v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities?options=keyValues&limit=50	GET	**url**:Link to the service that will be consulted.	Method to retrieve all entities of a service.
		port-orion: Port to connect with the service.	

Return:

```
All entities of the service and for each entity shows their information of compressed
↪way.
```

2.5.6 Update entity v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}/attrs	PATCH	**url**:Link to the service that will be consulted.	Method to update some attributes of the entity.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity where their attributes will be updated.	

Body:

```
{
    "attributeID":{
        value:"Attribute value",
        type:"Attribute type"
    }
}
```

Example of Body:

```
{
    "temperature": {
        "value": 26.5,
        "type": "Float"
    },
    "pressure": {
        "value": 763,
        "type": "Float"
    }
}
```

2.5.7 Delete entity v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}	DELETE	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity which will be deleted.	Method to delete an entity of a service.

2.5.8 Create attribute for entity v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}/attrs/	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity which will have new attributes.	Method to add a new attribute to the entity.

Body:

```
{
    "attributeID":{
        value:"Attribute value",
        type:"Attribute type"
    }
}
```

Example of Body:

```
{
    "temperature": {
        "value": 26.5,
        "type": "Float"
    }
}
```

2.5.9 Retrieve entity attribute v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}/attrs/{{fiware-attr}}	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity where is the attribute. **fiware-attr**: ID of the attribute.	Method to retrieve an attribute of an entity.

Return:

```
If the search has been successful then the attribute is retrieved and shows their␣
↪information else an error message is returned.
```

2.5.10 Retrieve entity attribute as data model v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}?options=keyValues	GET	**url**:Link to the service that will be consulted.	Method to retrieve an attribute of an entity.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity where is the attribute.	
		fiware-attr: ID of the attribute.	

Return:

```
If the search has been successful then the attribute is retrieved and shows their␣
↪information, else it returns a message error.
```

2.5.11 Delete attribute for entity v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/entities/{{fiware-entity}}/attrs/{{fiware-attr}}	DELETE	**url**:Link to the service that will be consulted.	Method to delete an attribute of an entity.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity which will delete attributes	
		fiware-attr: ID of the attribute which will delete	

Retrieve type v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/types/{{fiware-service}}	GET	**url**:Link to the service that will be consulted.	Method to retrieve an entity type of the service.
		port-orion: Port to connect with the service.	
		fiware-service: ID of type of the entity which will be searched in the service.	

Return:

```
If the search has been successful then type is retrieved and shows all their␣
↪information, else an error message is returned.
```

2.5.12 Retrieve types v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/types	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	The different entity types in the service and their information.

2.5.13 Create subscription v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/subscriptions	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to create a subscription to one or many entities.

Body:

```
{
    "description": "Definition of the subscription",
    "subject": {
        "entinties": [
            {
                "id": "Room1"
                "type: "Room"
            }
        ],
        "condition": {
            "attrs": [
                "pressure"
            ]
        }
    },
    "notification": {
        "http": {
            "url": "http://localhost:1028/accumulate"
        },
        "attrs": [
            "temperature"
        ]
    },
    "expires": "2040-01-01T14:00:00.00Z",
    "throttling": "5"
}
```

- **condition:** It defines the "trigger" for the subscription.
- **url:** URL where to send notifications
- **throttling:** It is used to specify a minimum inter-notification arrival time.
- **notification.attr:** Attributes which you will received in a notification when "condition.attr" changes.

2.5.14 Retrieve subscriptions v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/subscriptions	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to retrieve all subscriptions of a service.

Return:

```
All existing subscriptions of the service and for each subscription show their␣
↪information.
```

2.5.15 Remove subscriptions v2

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v2/subscriptions/{{fiware-subscription}}	DELETE	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-subscription**: ID of the subscription will be deleted.	Method to delete a subscription of a service.

2.5.16 Create context entity v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/registerContext	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to create a context of entity, an entity without values.

Body:

```
{
    "contextRegistrations": [
        {
            "entities": [
                {
                    "type": "entity_type",
                    "isPattern": "false",
                    "id": "entity_id"
                }
            ],
            "attributes": [
                {
                    "name": "nombre_atributo",
                    "type": "attribute_type",
                    "isDomain": "false"
                }
            ],
```

```
            "providingApplication": "http://homard.hopu.
            eu:1026/v2/entities"
        }
    ],
    "duration": "P1M"
}
```

- **isPattern:** Nowadays, it is not being used. Threfore, value is always "false".
- **isDomain:** The attribute domains aren't supported. Always 'false'.
- **providingApplication:** The URL that represents the context information of the registered entities and attributes.
- **duration:** The duration of the element. In ISO 8601 standard format.

Return:

```
Returns a confirmation that the item has been created correctly

{
    "duration": "P1M",
    "registrationId": "5a79812d777fc523840b8446"
}
```

2.5.17 Retrieve context entity v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/discoverContextAvailability	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to retrieve a context of entity, an entity without values, depends on ID and type.

Body:

```
{
    "entities": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": "entity_id"
        }
    ]
}
```

- **isPattern:** Nowadays, it is not being used. Threfore, value is always "false".

Return:

```
In case of finding this element, it returns the information.

In case of not finding this element, it returns a 404 error "No context element␣
↪found".
```

2.5.18 Create/update entity v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/updateContext	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to create or update an entity, depends on method's body.

Body (creation):

```
{
    "contextElements": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": "entity_id ",
            "attributes": [
                {
                    "name":"attribute_id ",
                    "type": "attribute_type",
                    "value": "attribute_value"
                }
            ]
        }
    ],
    "updateAction": "APPEND"
}
```

Body (updating):

```
{
    "contextElements": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": "entity_id ",
            "attributes": [
                {
                    "name":"attribute_id",
                    "type": "attribute_type",
                    "value": "attribute_value"
                }
            ]
        }
    ],
    "updateAction": "UPDATE"
}
```

- **isPattern:** Nowadays, it is not being used. Threfore, value is always "false".
- **updateAction:** Action to be carried out ("APPEND" or "UPDATE"). In case of "APPEND" creates the entity, if else "UPDATE" updatdes the entity.

Return:

```
In case of performing the method correctly, it return 200 OK together with the␣
↪information of the entity.
```

```
In UPDATE, in case of not find that entity, it return a 404 ERROR "No context element␣
↪found".
```

2.5.19 Create entity v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}	POST	**url**:Link to the service that will be consulted.	Method to create a new entity.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity that will be create of the service.	

Body:

```
{
    "type": "entity_type",
    "attributes": [
        {
            "name": "attribute_id",
            "type": "attribute_type",
            "value": "attribute_value"
        }
    ]
}
```

Return:

```
In case of performing the method correctly, it return 200 OK together with the␣
↪information of the new entity.
```

2.5.20 Retrieve entity standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}	GET	**url**:Link to the service that will be consulted.	Method that return a defined entity passed as parameter.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity that will be create of the service.	

Return:

```
In case of performing the method correctly, it return 200 OK together with the␣
↪information of the entity.

In case of not find that entity, it return a 404 ERROR "No context element found".
```

2.5.21 Retrieve entity as object standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}?attributeFormat=object	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity that will be create of the service.	Method that return a defined entity with object JSON format.

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of the entity with object JSON format.

In case of not find that entity, it return a 404 ERROR "No context element found".
```

2.5.22 Retrieve entity convenience v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/queryContext	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method that returns a defined entity passed in the body of the method.

Body:

```
{
    "entities": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": "entity_id"
        }
    ]
}
```

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of the entity.

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context␣
↪element found".
```

2.5.23 Retrieve entities as object convenience v1

URL	Method	URL Params	Definition
http://{{url}}:{{portorion}}/v1/queryContext?attributeFormat=object	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method that return a defined entity with object JSON format. The entity ID is passed in the body of method.

Body:

```
{
    "entities": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": "entity_id"
        }
    ]
}
```

Return:

```
In case of performing the method correctly, it returns 200 OK together with the
↪information of the entity with object JSON format.

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context
↪element found".
```

2.5.24 Retrieve entities v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method that return all entities.

Return:

```
Show all existing entities.
```

2.5.25 Retrieve entities as object v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities?attributeFormat=object	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method that return all entities with object JSON format.

Return:

```
Show all existing entities with object JSON format.
```

2.5.26 Retrieve entities for type v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/contextEntityTypes/{{fiware-type}}	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-type**: Type of entities that we want to receive.	Method that returns all entities of a concrete type.

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of the all entities of this type.

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context␣
↪element found".
```

2.5.27 Retrieve entities for type as object v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/contextEntityTypes/{{fiware-type}}?attributeFormat=object	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-type**: Type of entities that we want to receive.	Method that return all entities of a concrete type with object JSON format.

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of the all entities of this type with object JSON format.

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context␣
↪element found".
```

2.5.28 Delete entity v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}	DELETE	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity we want to delete.	Method to delete a defined entity.

Return:

```
In case of performing the method correctly and delete the entity defined, it returns␣
↪200 OK.

In case of not finding that entity, it returns a 404 ERROR "No context element found".
```

2.5.29 Create/update entity attribute v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}/attributes	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity that will be created of the service.	Method to update the values of one or many attributes of an entity passed as parameter. In addition, this method can create new attributes for an entity.

Body (creation):

```
Create a new attribute. We transmit a new attribute in the body:

{
    "attributes": [
        {
            "name": "attribute_id",
            "type": "attribute_type",
            "value": "attribute_value"
        }
    ]
}
```

Body (updating):

```
Update an attribute. We transmit an existing/available attribute with a new value.

{
    "attributes": [
        {
            "name": "attribute_id_new",
            "type": "attribute_type_new",
            "value": "attribute_value_new"
```

```
        }
    ]
}
```

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of the entity updating.

In case of not finding that entity, it returns a 404 ERROR "No context element found".
```

2.5.30 Retrieve entity attribute v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}/attributes/{{fiware-attr}}	GET	**url**:Link to the service that will be consulted.	Method that retrieves the defined attribute depending on the ID of a specific entity.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity that will be retrieved of the service.	
		fiware-attr: Attribute ID we want to receive.	

Return:

```
In case of performing the method correctly, it returns 200 OK together the attribute␣
↪value defining by its ID for a specific entity.

In case of not finding that entity or attribute, it returns 404 ERROR "No context␣
↪element found".

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context␣
↪element found".
```

2.5.31 Delete entity attribute as object v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}/attributes/{{fiware-attr}}?attributeFormat=object	GET	**url**:Link to the service that will be consulted.	Method that retrieves the defined attribute depending on the ID of a specific entity with object JSON format.
		port-orion: Port to connect with the service.	
		fiware-entity: ID of the entity that will be retrieved of the service.	
		fiware-attr: Attribute ID we want to receive.	

Return:

```
In case of performing the method correctly, it returns 200 OK together the attribute␣
↪value defined by its ID for a specific entity with object JSON format.

In case of not finding that entity or attribute, it returns 404 ERROR "No context␣
↪element found".
```

2.5.32 Retrieve entities attribute for type v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/contextEntityTypes/{{fiware-type}}/attributes/{{fiware-attr}}	GET	**url**:Link to the service that will be consulted.	Method that returns the defined attribute of all entities of a specific type.
		port-orion: Port to connect with the service.	
		fiware-type: Type of entities that we want to receive.	
		fiware-attr: Attribute of type of entities that we want to receive.	

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of defined attributes of all entities of this type.

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context␣
↪element found".
```

2.5.33 Retrieve entities attribute for type as object v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/contextEntityTypes/{{fiware-type}}/attributes/{{fiware-attr}}?attributeFormat=object	GET	**url**:Link to the service that will be consulted.	Method that returns the defined attribute of all entities of a specific type with object JSON format.
		port-orion: Port to connect with the service.	
		fiware-type: Type of entities that we want to receive.	
		fiware-attr: Attribute of type of entities that we want to receive.	

Return:

```
In case of performing the method correctly, it returns 200 OK together with the␣
↪information of defined attributes of all entities of this type with object JSON␣
↪format.

In UPDATE, in case of not finding that entity, it returns a 404 ERROR "No context␣
↪element found".
```

2.5.34 Delete entity attribute v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextEntities/{{fiware-entity}}/attributes/{{fiware-attr}}	DELETE	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-entity**: ID of the entity that will be retrieve of the service. **fiware-attr**: Attribute ID we want to receive.	Method to delete the defined attribute depending on the ID of a concrete entity.

Return:

```
In case of performing the method correctly and delete the entity defined, it returns␣
↪200 OK.

In case of not finding that entity or attribute, it returns 404 ERROR "No context␣
↪element found".
```

2.5.35 Retrieve type v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextTypes/{{fiware-type}}	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-type**: Type of entities that we want to receive.	Method to receive an entity type concrete.

Return:

```
In case of performing the method correctly, it returns 200 OK together the␣
↪information of this entities type.

In case of not finding that entities type, it returns 404 ERROR "No context element␣
↪found".
```

2.5.36 Retrieve types v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextTypes	GET	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to receive the existing entities types.

Return:

```
In case of performing the method correctly, it returns 200 OK together the␣
↪information of all entities type.

In case of not finding that entities type, it returns 404 ERROR "No context element␣
↪found".
```

2.5.37 Create context subscription standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/subscribeContextAvailability	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to create a subscription to one or many context entities.

Body:

```
{
    "entities": [
        {
            "type": "id_entity_type",
            "isPattern": "false",
            "id": ".*"
        }
    ],
    "attributes": [
        "id_attribute"
    ],
    "reference": "http://cygnus:5050/notify",
    "duration": "P1M"
}
```

- **isPattern:** Nowadays, it is not being used. Threfore, value is always "false".
- **id:** To which entity it wants to subscribe. In this case, to all entities of those type.
- **reference:** URL of client that it want to subscribe.
- **duration:** The duration of the subscription. In ISO 8601 standard format.

Return:

```
{
    "subscriptionId": "subscription_id",
    "duration": "P1M"
}
```

2.5.38 Update context subscription standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/updateContextAvailabilitySubscriptions	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to update a subscription of a context entity.

Body:

```
{
    "entities": [
        {
            "type": "id_entity_type",
            "isPattern": "false",
            "id": ".*"
        }
    ],
    "duration": "P1M"
    "subscriptionId": "subscription_id"
}
```

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "id_parameter": "value_parameter"
    }
}
```

2.5.39 Update context subscription convenience v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/registry/contextAvailabilitySubscriptions/{{fiware-subscription}}	PUT	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-subscription**: ID of the subscription will be updated.	Method to update a subscription of a context entity.

Body:

```
{
     "entities": [
         {
             "type": "id_entity_type",
             "isPattern": "false",
             "id": ".*"
         }
     ],
     "duration": "P1M"
     "subscriptionId": "subscription_id"
}
```

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "id_parameter": "value_parameter"
```

```
    }
}
```

2.5.40 Delete context subscription standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/unsuscribeContextAvailability	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to delete a subscription of a context entity.

Body:

```
{
     "subscriptionId": "5a785788777fc523840b843e"
}
```

Return:

```
In case of performing the method correctly and delete the entity defined, it returns␣
↪200 OK together with the removed subscription ID.

In case of not finding that subscription, it returns 404 ERROR "No context element␣
↪found".
```

2.5.41 Delete context subscription convenience v1

URL	Method	URL Params	Definition
http://{{url}}:{{portorion}}/v1/contextAvailabilitySubscriptions/{{fiware-subscription}}	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-subscription**: ID of subscription to remove.	Method to delete a subscription of a context entity.

Return:

```
In case of performing the method correctly and delete the entity defined, it returns␣
↪200 OK together with the removed subscription ID.

In case of not finding that subscription, it returns 404 ERROR "No context element␣
↪found".
```

2.5.42 Create subscription standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/suscribeContext	POST	**url**:Link to the service that will be consulted.	Method to create a subscription to one or many entities.
		port-orion: Port to connect with the service.	

Body:

```
{
    "entities": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": ".*"
        }
    ],
    "attributes": [
        "attribute_id"
    ],
    "reference": "http://cygnus:5050/notify",
    "duration": "P1M"
    "notifyConditions": [
        {
            "type": "ONCHANGE",
            "condValues": [
                "attribute_id"
            ]
        }
    ],
    "throttling": "PT5S"
}
```

- **isPattern:** Currently is hasn't use. Always 'false'.
- **id:** To which entity it wants to subscribe. In this case, to all entities of those type.
- **reference:** URL of client that it want to subscribe.
- **duration:** The duration of the subscription. In ISO 8601 standard format.
- **notifyConditions:** Define the launcher to notify the subscriptions. In this case, when change a value ofattributes transmitted via the "condValues" of an entity ,then it will trigger a notification.
- **throttling:** Specify a minimum inter-notification arrival time. In this case, 5 seconds.

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "duration": "P1M"
    }
}
```

2.5.43 Create subscription convenience v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextSubscriptions	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to create a subscription to one or many entities.

Body:

```
{
    "entities": [
        {
            "type": "entity_type",
            "isPattern": "false",
            "id": ".*"
        }
    ],
    "attributes": [
        "attribute_id"
    ],
    "reference": "http://cygnus:5050/notify",
    "duration": "P1M"
    "notifyConditions": [
        {
            "type": "ONCHANGE",
            "condValues": [
                "attribute_id"
            ]
        }
    ],
    "throttling": "PT5S"
}
```

- **isPattern:** Currently is hasn't use. Always 'false'.
- **id:** To which entity it wants to subscribe. In this case, to all entities of those type.
- **reference:** URL of client that it want to subscribe.
- **duration:** The duration of the subscription. In ISO 8601 standard format.
- **notifyConditions:** Define the launcher to notify the subscriptions. In this case, when change a value ofattributes transmitted via the "condValues" of an entity ,then it will trigger a notification.
- **throttling:** Specify a minimum inter-notification arrival time. In this case, 5 seconds.

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "duration": "P1M"
    }
}
```

2.5.44 Update subscription standard v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/updateContextSubscription	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service.	Method to update a subscription of an entity.

Body:

```
{
    "subscriptionId": "id_subscription",
    "id_parameter": "value_parameter"
}
```

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "id_parameter": "value_parameter"
    }
}
```

2.5.45 Update subscription convenience v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextSubscriptions/{{fiware-subscription}}	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-subscription**: ID of subscription to update.	Method to update a subscription of an entity.

Body:

```
{
    "subscriptionId": "id_subscription",
    "id_parameter": "value_parameter"
}
```

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "id_parameter": "value_parameter"
    }
}
```

2.5.46 Delete subscription v1

URL	Method	URL Params	Definition
http://{{url}}:{{port-orion}}/v1/contextSubscriptions/{{fiware-subscription}}	POST	**url**:Link to the service that will be consulted. **port-orion**: Port to connect with the service. **fiware-subscription**: ID of subscription to remove.	Method to delete the subscription to an entity.

Return:

```
{
    "subscribeResponse": {
        "subscriptionId": "id_subscription",
        "id_parameter": "value_parameter"
    }
}
```

2.6 ANNEX 1: OMA LwM2M

OMA LightweightM2M is a device management protocol designed for sensor networks and the demands of a machine-to-machine (M2M) environment. With LwM2M, OMA has responded to demand in the market for a common standard for managing lightweight and low power devices on a variety of networks necessary to realize the potential of IoT. The LwM2M protocol, designed for remote management of M2M devices and related service enablement, features a modern architectural design based on REST, defines an extensible resource and data model and builds on an efficient secure data transfer standard called the Constrained Application Protocol (CoAP). LwM2M has been specified by a group of industry experts at the Open Mobile Alliance's Device Management Working Group and is based on protocol and security standards from the IETF.

More information about OMA LwM2M Protocol: http://www.openmobilealliance.org/release/LightweightM2M/V1_0_1-20170704-A/OMA-TS-LightweightM2M-V1_0_1-20170704-A.pdf

2.6.1 Resource model

The LwM2M Enabler defines a simple resource model where each piece of information made available by the LwM2M Client is a Resource. Resources are organized into Objects, and each Resource is given a unique identifier within that Object.

Each Objec is assigned a unique OMA LwM2M Object identifier allocated and maintained by the OMA Naming Authority (OMNA). Further Objects may be added by OMA or other organizations to enable additional M2M Services.

As an Object only specifies a grouping of Resources, an Object must be firstly instantiated so that the LwM2M Client can use the Resources of such an Object and the associated functionalities.

When an Object is instantiated an Object Instance is created with a subset of the Resources defined in the Object specification; a LwM2M Server can then access that Object Instance and its set of instantiated Resources.

A Resource which is instantiated within an Object Instance is a Resource which can either:

- contain a value (if the Resource is Readable and/or Writeable)
- or can be addressed by a LwM2M Server to trigger an action in the LwM2M Client (if the Resource is Executable)

The Object specification defines the operations (Read, Write, Execute) which are individually supported by the Resources belonging to that Object; this specification also defines the Mandatory or Optional characteristics of such Resources.

Objects and Resources have the capability to have multiple instances. Multiple-Instances Resources can be instantiated by LwM2M Server operations in using JSON or TLV formats. The LwM2M Client also has the capability to instantiate Single or Multiple-Instances Resources.

The LwM2M Enabler defines an access control mechanism per Object Instance. Object Instances should have an associated Access Control Object Instance. An Access Control Object Instances contains Access Control Lists (ACLs) that define which operations on a given Object Instance are allowed for which LwM2M Server(s).

2.6.2 Objects

Each Object definition, which may be found in the LwM2M specification, features the following information:

- **Name:** specifies the Object name.
- **Object ID:** specifies the Object ID.
- **Instances:** indicates whether this Object supports multiple Object Instances or not. If this field is "Multiple" then the number of Object Instance can be from 0 to many. If this field is "Single" then the number of Object Instance can be from 0 to 1.
- **Mandatory:** if this field is "Mandatory", then the LwM2M Client MUST support this Object. If this field is "Optional", then the LwM2M Client SHOULD support this Object.
- **Object URN**
- **Resource definitions**

2.6.3 Attributes

Attributes are metadata which can be attached to an Object, an Object Instance or a Resource. The value of an Attribute is LwM2M Server specific.

Regardless to the LwM2M entity a given Attribute is attached to, the value of such an Attribute can be set at various levels: Object, Object Instance, Resource levels.

- **<PROPERTIES> Class Attributes: The role of these Attributes is to provide metadata which may communicate helpful in**
  - **Dimension** (dim): Number of Instantiations for a Multiple Resource.
  - **Object Version** (ver): Provide the version of the associated Object.
- **<NOTIFICATION>** Class Attributes: The role of these R-Attributes is to provide parameters to the "Notify" operation; any readable Resource can have such Rattributes. In the message sent by a LwM2M Client in response to an "Observe" operation, the current Resource value is reported; this event can be considered as the initial notification.

  Each time a Resource notification is sent, the "Minimum Period" and "Maximum Period" timers associated to this Resource are restarted.

  The notification of a Resource value will be sent when the combination of a change value condition ("Greater Than", "Less Than", or "Step") and the "Minimum Period" timing conditions are both fulfilled for that Resource.

  This behaviour can be modified using the following available attributes:

- **Minimum Period** (pmin): The Minimum Period Attribute indicates the minimum time in seconds the LwM2M Client must wait between two notifications.
- **Maximum Period** (pmax): The Maximum Period Attribute indicates the maximum time in seconds the LwM2M Client MAY wait between two notifications.
- **Greater Than** (gt) and Less Than (lt): This Attributes defines a threshold high value and low value. When these Attributes is present, the LwM2M Client must notify the Server each time the Observed Resource value crosses these thresholds with respect to pmin parameter and to pmax parameter.
- **Step** (st): This Attribute defines a minimum change value between two notifications. When this Attribute is present, the change value condition will occur when the value variation since the last notification of the Observed Resource, is greater or equal to the "Step" Attribute value.

2.6.4 Clients and servers

This enabler defines the application layer communication protocol between a LwM2M Server and a LwM2M Client, which is located in a LwM2M Device. The OMA Lightweight M2M enabler includes device management and service enablement for LwM2M Devices. The target LwM2M Devices for this enabler are mainly resource constrained devices. Therefore, this enabler makes use of a light and compact protocol as well as an efficient resource data model.

A Client-Server architecture is introduced for the LwM2M Enabler, where the LwM2M Device acts as a LwM2M Client and the M2M service, platform or application acts as the LwM2M Server. The LwM2M Enabler has two components, LwM2M Server and LwM2M Client. Four interfaces are designed between these two components:

- Bootstrap
- Client Registration
- Device management and service enablement
- Information Reporting

2.6.5 Bootstrap Interface

The Bootstrap Interface is used to provision essential information into the LwM2M Client to enable the LwM2M Client to perform the operation "Register" with one or more LwM2M Servers.

During the Bootstrap Phase, the Client may ignore requests and flush all pending responses not related to the Bootstrap sequence. There are four bootstrap modes supported by the LwM2M Enabler:

- Factory Bootstrap
- Bootstrap from Smartcard
- Client Initiated Bootstrap
- Server Initiated Bootstrap

The last two Bootstrap modes require the help of a LwM2M Bootstrap-Server to achieve the ultimate goal to connect a LwM2M Client to their LwM2M Server(s).

The LwM2M Client must support at least one bootstrap mode specified in the Bootstrap Interface.

The LwM2M Bootstrap-Server must support Client Initiated Bootstrap and Server Initiated Bootstrap modes specified in the Bootstrap Interface.

2.6.6 Client Registration Interface

The LwM2M Server must support all the operations in this interface and the LwM2M Client must support "Register" and "Update" and should support "De-register" operation.

The Client Registration Interface is used by a LwM2M Client to register with one or more LwM2M Servers, maintain each registration and de-register from a LwM2M Server. The registration is based on the Resource Model and Identifiers defined in Section 6 Identifiers and Resources. When registering, the LwM2M Client performs the "Register" operation and provides the properties the LwM2M Server requires to contact the LwM2M Client (e.g., End Point Name); maintain the registration and session (e.g., Lifetime, Queue Mode) between the LwM2M Client and LwM2M Server as well as knowledge of the Objects the LwM2M Client supports and existing Object Instances in the LwM2M Client. The registration is soft state, with a lifetime indicated by the Lifetime Resource of that LwM2M Server Object Instance. The LwM2M Client periodically performs an update of its registration information to the registered LwM2M Server(s) by performing the "Update" operation– possibly without any parameters. If the lifetime of a registration expires without receiving an update from the LwM2M Client:

- The LwM2M Server must remove the registration of that Client.
- The LwM2M Client must re-register ("Update" is not sufficient) to the LwM2M Server in order to be connected again, before initiating any further communication.

If the LwM2M Server or the LwM2M Client set a value to the Lifetime Resource of the Server Object Instance, this value becomes the new lifetime of the Registration session.

During "Register" or "Update" Operations, the parameter Lifetime – if present – must match the current value of the Mandatory Lifetime Resource of the LwM2M Server Object Instance. Finally, when shutting down or discontinuing use of a LwM2M Server, the LwM2M Client performs a "De-register" operation.

The Binding Resource of the LwM2M Server Object informs the LwM2M Client of the transport protocol preferences of the LwM2M Server for the communication session between the LwM2M Client and LwM2M Server.

The LwM2M Client should perform the operations with the modes indicated by the Binding Resource of the LwM2M Server Object Instance.

2.6.7 Device Management and Service Enablement Interface

The LwM2M Server and the LwM2M Client must support all the operations in this interface.

The Device Management and Service Enable Interface is used by the LwM2M Server to access Object Instances and Resources available from a registered LwM2M Client. The interface provides this access through the use of "Create", "Read", "Write", "Delete", "Execute", "Write-Attributes", or "Discover" operations.

The Device Management and Service Enablement interface defines the following commands:

- **Read** operation is used to access the value of a Resource, an array of Resource Instances, an Object Instance or all the Object Instances of an Object.
- **Discover** operation is used to discover LwM2M Attributes attached to an Object, Object Instances, and Resources. This operation can be used to discover which Resources are instantiated in a given Object Instance.
- **Write** operation is used to change the value of a Resource, the values of an array of Resources Instances or the values of multiple Resources from an Object Instance.
- **Write-Attributes:** In LwM2M 1.0, only Attributes from the **<NOTIFICATION>** class may be changed in using the "Write-Attributes" operation. The operation permits multiple Attributes to be modified within the same operation.
- **Execute** operation is used by the LwM2M Server to initiate some action, and can only be performed on individual Resources. A LwM2M Client must return an error when the "Execute" operation is received for an Object Instance(s) or Resource Instance(s).

- **Create** operation is used by the LwM2M Server to create Object Instance(s) within the LwM2M Client. The "Create" operation must target an Object. If any error occurs, nothing must be created.
- Finally, the **Delete** operation is used for LwM2M Server to delete an Object Instance within the LwM2M Client.

The Object Instance that is deleted in the LwM2M Client by the LwM2M Server must be an Object Instance that is announced by the LwM2M Client to the LwM2M Server using the "Register" and "Update" operations of the Client Registration Interface.

2.6.8 Information Reporting Interface

The LwM2M Server and the LwM2M Client must support all the operations in this interface.

The Information Reporting Interface is used by a LwM2M Server to observe any changes in a Resource on a registered LwM2M Client, receiving notifications when new values are available. This observation relationship is initiated by sending an "Observe" operation to the LwM2M Client for an Object, an Object Instance or a Resource. An observation ends when a "Cancel Observation" operation is performed.

- **Observe:** The LwM2M Server initiates an observation request for changes of a specific Resource, Resources within an Object Instance or for all the Object Instances of an Object within the LwM2M Client.
- **Notify:** The "Notify" operation is sent from the LwM2M Client to the LwM2M Server during a valid observation on an Object Instance or Resource. This operation includes the new value of the Object Instance or Resource. The "Notify" operation should be sent when all the conditions (i.e., Minimum Period, Maximum Period, Greater Than, Less Than, Step) configured by "Write-Attributes" operation for "Observe" operation are met.
- **Cancel Observation:** The "Cancel Observation" operation is sent from the LwM2M Server to the LwM2M Client to end an observation relationship for Object Instance or Resource.

2.7 ANNEX 2: FIWARE-DOCKER Architecture

FIWARE IoT Stack allows you to connect devices and receive data. Though to FIWARE IoT Stack we can understand, interpret and process the data relative to physical devices. In this document, we have exposed and defined the components of FIWARE IoT Stack we have used in a summarized fashion.

2.7.1 MongoDB

MongoDB is a free and open-source database. It is classified as a NoSQL database. Instead of save the data on tables like is done in relational databases, MongoDB save data structure on documents similar to JSON with a dynamic diagram, making that the data integration on certain application be more easy and quickly.

This database is connected with:

- STH Comet save aggregated data.
- Cygnus save raw data.
- Orion Context Broker save subscriptions.
- IoT Agent save device mapping.

More information about the component can be found in the MongoDB documentation: https://docs.mongodb.com

2.7.2 STH-Comet

The Short Time Historic (STH) is a component of the FIWARE in charge of manage historical time series information about the evolution of context data registered in an Orion Context Broker instance over time. STH can be considered as a Time Series Database. This component has a main functionality:

- Retrieves, store, update or delete raw and aggregated time series context information.

Also, STH is connected to:

- MongoDB is in charge of store the desired raw and aggregated time series context information managed by the STH component.
- Cygnus is needs to store the raw and aggregated time series context information into the STH component following the formal option, because Cygnus is in charge of store the context

information managed by an Orion Context Broker instance over time. In other words, Cygnus notify to STH-Comet with data received.

More information about the component can be found in the Fiware-STH-Comet documentation: https://fiware-sthcomet.readthedocs.io/en/latest/index.html

2.7.3 Cygnus

Cygnus is based on Apache Flume, a distributed service for collecting, aggregating and moving data. Cygnus is a connector in charge of persisting Orion context data. Store the Orion Context Broker entities data, allows create a historical view of this data. Cygnus subscribes to Orion and receive entities data when this entities are modified. Cygnus defined what data it is interested in an Orion entity. Cygnus allows:

- Store the last value of an entity's attribute. If and older value is required, Cygnus receives it and sends to the applicant.

Cygnus is connected to:

- STH-Comet. Cygnus notify to STH-Comet when a data received. A STH database built on top of MongoDB.
- Orion. Cygnus is subscribed to Orion Context Broker for receive data about entities it interesting.
- MongoDB. Cygnus store the context data in a MongoDB database.

More information about the component can be found in the Cygnus documentation: http://fiware-cygnus.readthedocs.io/en/1.7.1/index.html

2.7.4 Orion Context Broker

Orion is a C++ implementation developed as a part of FIWARE platform. Orion Context Broker is in charge of manage all the lifecycle of context information including update, queries, registrations and delete. Orion allows subscribe to a context information for notify when any modify occur and create and modify entities through queries. Currently coexist the NGSIv1 version and the NGSIv2.

Orion Context Broker allows:

- Manage of entities. OCB is in charge of create, update, retrieve and delete an entity or list of entities. Also, allows manage entities by their type.
- Manage of context information. OCB allows create, update and delete a context information.
- Manage of subscription. OCB is in charge of create, update, retrieve and delete a subscription.

Also, Orion Context Broker depends on:

- MongoDB. This database store the subscriptions they want to be notified when modify occur.
- IoT Agent. IoT Agent is subscribed to Orion Context Broker for receive data about entities it interesting.
- Cygnus. Cygnus is subscribed to Orion Context Broker for receive data about entities it interesting.
- QuantumLeap. QuantumLeap is subscribed to Orion Context Broker for receive data about entities it interesting.
- Perseo-fe. Perseo-fe is subscribed to Orion Context Broker for receive data about entities it interesting.

More information about the component can be found in the Orion Context Broker documentation: https://fiware-orion.readthedocs.io/en/master/index.html

2.7.5 IoT Agent

IoT Agent it's a component in charge on stores devices data. IoT Agent receive data in M2M language and then it translate to standard platform language to can be processed for Orion Context Broker. In other words, IoT Agent converts the physical devices to an Orion Context Broker entity. Also, IoT Agent allows to send commands to devices. IoT Agents supports OMA lightweight M2M protocol and JSON protocol.

IoT Agent has various functionalities:

- Transforms the physical devices to an entities it can be manage in Orion Context Broker.
- IoT Agent take part between OMA LWM2M and JSON Objects.
- Store the translation of a physical device in a persistent storage.

Also, IoT Agent is connected to:

- Orion. IoT Agent send the JSON objects representing physical devices to Orion.
- MongoDB. Store the mapping of the physical device in a MongoDB database.

More information about the component can be found in the IoT Agent for OMA LWM2M documentation: http://fiwareiotagent-lwm2m.readthedocs.io/en/latest/index.html

2.7.6 Grafana

Grafana is an open source application allows generate dashboards from data contained in CrateDB database. The CrateDB data is displayed in a dashboards, and the user can display the data the way his want it. However, Orion Context Broker store the information in MongoDB database. To store the relevant Orion Context Information data in a CrateDB database we need QuantumLeap to convert the OCB data to CrateDB data.

Grafana has a main functionality:

- Show the values of the Orion Context Broker entities as a graph order by data time.

Grafana is connected to:

- CrateDB. Grafana shows data belonging to CrateDB database.
- QuantumLeap. Convert the storage of NGSI FIWARE NGSIv2 data into a time series database for CrateDB.

More information about the component can be found in the Grafana experiments: https://github.com/smartsdk/ngsitimeseries-api/tree/master/experiments/grafana

2.7.7 QuantumLeap

QuantumLeap is an API that supports the storage of NGSI FIWARE NGSIv2 data into a time series database. The client create an Orion subscription to notify QuantumLeap of the update in the entities it interesting. QuantumLeap translate the NGSI information receive to Orion subscription to CrateDB database, and persisting it.

QuantumLeap allows:

- Convert the Orion Context Broker information to CrateDB data.
- Supported integration with dashboard application like Grafana.

QuantumLeap is connected to:

- CrateDB. QuantumLeap allows to CrateDB store the Orion Context Broker entities data.
- Orion Context Broker. QuantumLeap is subscribed to Orion Context Broker for receive data about entities it interesting.

More information about the component can be found in the QuantumLeap documentation: https://smartsdk.github.io/ngsi-timeseries-api/

2.7.8 CrateDB

CrateDB is a distributed SQL database in charge to store and analyze machine data in real-time. Crate is the database where NGSI data will be persisted after its converts to a data time series. Once the Orion subscription of QuantumLeap is created, when QuantumLeap receives NGSI information, coverts it and stores this information in a CrateDB database.

CrateDB has various functionalities:

- Stores the data as a data time series.
- Allows show the data it contained with dashboards application.

CrateDB is connected to:

- QuantumLeap. CrateDB is the QuantumLeap's backend where NGSI data converted will be stored.
- Grafana. Grafana generate dashboards from data of CrateDB database, it obtains thanks to QuantumLeap.

More information about the component can be found in the CrateDB documentation: https://crate.io/docs/clients/admin-ui/en/latest/index.html

2.7.9 Perseo-core

Perseo-core is the rules engine of Perseo CEP, the "back-end". It receives the events through a POST a JSON object and apply the functions generate by this JSON object. In case of must to execute an action, it informs to Perseo-fe through a POST request. EPL is the language of rules of Esper. Esper is a Java library in charge of event processing logic and the rules engine. The defined rules aren't stored persistently, but kept in memory, and it can update from Perseo-fe.

Perseo-core has the following functionality:

- Perseo-core trigger events through a POST request to Perseo-fe if any rule is complied it.
- Acts as the rules engine and event process logic of Perseo CEP.

Perseo-core is connected to:

- Perseo-fe. Perseo-fe send to perseo-core a rule and event, Perseo-core process the information and, if any rule is complied it, send an action to Perseo-fe.

More information about the component can be found in the Perseo-core documentation: https://github.com/telefonicaid/perseo-core/tree/master/documentation

2.7.10 Perseo-fe

Perseo-fe is a component that together with Perseo-core form the Perseo CEP, the "front-end". It processing incoming events a rules, storing rules and executing actions. When Perseo-core send an action to Perseo-fe, it is responsible of send an action via SMS, e-mail o HTTP. We must have configured the following servers if we can send a notification: SMPP, SMTP and HTTP.

Perseo-fe allows:

- Send a SMS, e-mail, HTT Post or a tweet if it receive from Perseo-fe any action.
- Receives events of creation or update of Orion Context Broker entities.
- Store the rules to check when it receive an events.

Perseo-fe is connected to:

- Orion. Perseo-fe is subscribed to Orion Context Broker for receive data about entities it interesting.
- Perseo-core. Perseo-core is the event processing logic and the rules engine of Perseo-fe, his brain.

More information about the component can be found in the Perseo-fe documentation: https://github.com/telefonicaid/perseo-fe/tree/master/documentation

2.8 ANNEX 3: STH-Comet RESTFul API

2.8.1 Retrieve raw values with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?hLimit={{fiware-limit}}&hOffset={{fiware-offset}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-limit**: Entries number for each page. **fiware-offset**: The offset to apply to the requested search of raw context information.	Method to retrieve all changes of an attribute.

Return:

```
From all registered changes of the attribute will choose the number of changes␣
↪specified in "fiware-limit" and depending on the value of "fiware-offset" will␣
↪return certain changes.
```

2.8.2 Retrieve raw values with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-filetype**: File format will be saved the changes of the attribute. Currently, the only supported value is "csv".	Method to retrieve all changes of an attribute.

Return:

```
Setting file with the number of changes of the attribute specified in "fiware-limit"␣
↪and from all changes registered, certain changes are chosen depending on the value␣
↪of "fiware-offset".
```

2.8.3 Retrieve n raw values

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?lastN={{fiware-lastN}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" changes of an attribute.

Return:

```
The last "n" changes of the attribute.
```

2.8.4 Retrieve n raw values with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?hLimit={{fiware-limit}}&hOffset={{fiware-offset}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" changes of an attribute.

Return:

```
From last "n" registered changes of the attribute will choose the number of changes␣
↪specified in "fiware-limit" and depending on the value of "fiware-offset" will␣
↪return certain changes.
```

2.8.5 Retrieve n raw with filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-lastN**: Entries number will be returned. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value is "csv".	Method to retrieve the last "n" changes of an attribute.

Return:

```
Setting file with the last "n" registered changes of the attribute.
```

2.8.6 Retrieve n raw values with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-lastN**: Entries number will be returned. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value is "csv".	Method to retrieve the last "n" changes of an attribute.

Return:

```
Setting file with the number of changes of the attribute specified in "fiware-limit"␣
↪and from the last "n" changes registered certain changes are chosen depending on␣
↪the value of
"fiware-offset".
```

2.8.7 Retrieve raw values between two dates with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number.	Method to retrieve all changes of an attribute between two dates.

Return:

```
From all registered changes of the attribute between the two specified dates will␣
↪choose the number of changes specified in "fiware-limit" and depending on the value␣
↪of
"fiware-offset" will return certain changes.
```

2.8.8 Retrieve n raw values between two dates

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}&lastN={{fiware-lastN}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" changes of an attribute between two dates.

Return:

```
The last "n" registered changes of the attribute between the two specified dates.
```

2.8.9 Retrieve raw value between two dates with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is “csv”.	Method to retrieve all changes of an attribute between two dates.

Return:

```
Setting file with the number of changes of the attribute specified in “fiware-limit”␣
↪and from all changes registered between the two specified dates, certain changes␣
↪are chosen
depending on the value of “fiware-offset”.
```

2.8.10 Retrieve n raw values between two dates with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}&lastN={{fiware-lastN}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number.	Method to retrieve the last "n" changes of an attribute between two dates.

Return:

```
From the last "n" registered changes of the attribute between the two specified dates␣
↪will choose the number of changes specified in "fiware-limit" and depending on the␣
↪value of
"fiware-offset" will return certain changes.
```

2.8.11 Retrieve n raw values between two dates with filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}&lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is "csv".	Method to retrieve the last "n" changes of an attribute between two dates.

Return:

```
Setting file with the last "n" registered changes of the attribute between the two␣
↪specified dates.
```

2.8.12 Retrieve n raw values between two dates with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}&lastN={{fiware-lastN}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is “csv”.	Method to retrieve the last “n” changes of an attribute between two dates.

Return:

```
Setting file with the number of changes of the attribute specified in “fiware-limit”␣
↪and from the last “n” changes registered between the two specified dates, certain␣
↪changes
are chosen depending on the value of “fiware-offset”.
```

2.8.13 Retrieve raw values after a date with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number.	Method to retrieve all changes of an attribute after a date.

Return:

```
From all registered changes of the attribute after the specified date will choose the␣
↪number of changes specified in "fiware-limit" and depending on the value of
"fiware-offset" will return certain changes.
```

2.8.14 Retrieve n raw values after date with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&lastN={{fiware-lastN}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" of an attribute after a date.

Return:

```
From the last "n" registered changes of the attribute after the specified date will␣
↪choose the number of changes pecified in "fiware-limit" and depending on the value␣
↪of
"fiware-offset" will return certain changes.
```

2.8.15 Retrieve raw values after date with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-lastN**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is "csv".	Method to retrieve all changes of an attribute after a date.

Return:

```
Setting file with the number of changes of the attribute specified in "fiware-limit"␣
↪and from all changes registered after the specified date, certain changes are chosen
depending on the value of "fiware-offset".
```

2.8.16 Retrieve n raw values after date

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&lastN={{fiware-lastN}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" of an attribute after a date.

Return:

```
The last "n" registered changes of the attribute and after the specified date.
```

2.8.17 Retrieve n raw values after date with filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-lastN**: Entries number will be returned. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is "csv".	Method to retrieve the last "n" of an attribute after a date.

Return:

```
Setting file with the last "n" registered changes of the attribute after the␣
↪specified date.
```

2.8.18 Retrieve n raw values after date with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateFrom={{fiware-dateFrom}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateFrom**: The starting date and time from which the raw context information is desired. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" of an attribute after a date.

Return:

```
Setting file with the number of changes of the attribute specified in "fiware-limit"␣
↪and from the last "n" changes registered after the specified date, certain
changes are chosen depending on the value of "fiware-offset".
```

2.8.19 Retrieve n raw values before date with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateTo={{fiware-dateTo}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is "csv".	Method to retrieve the last "n" registered changes before a date.

Return:

```
Setting file with the number of changes of the attribute specified in "fiware-limit"␣
↪and from the last "n" changes registered before the specified date, certain changes␣
↪are
chosen depending on the value of "fiware-offset".
```

2.8.20 Retrieve n raw values before date with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateTo={{fiware-dateTo}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&lastN={{fiware-lastN}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number.	Method to retrieve the last "n" registered changes before a date.

Return:

```
From the last "n" registered changes of the attribute before the specified date will␣
↪choose the number of changes specified in "fiware-limit" and depending on the value␣
↪of
"fiware-offset" will return certain changes.
```

2.8.21 Retrieve n raw values before date with filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateTo={{fiware-dateTo}}&lastN={{fiware-lastN}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is "csv".	Method to retrieve the last "n" registered changes before a date.

Return:

```
Setting file with the last "n" registered changes of the attribute before the␣
↪specified date.
```

2.8.22 Retrieve n raw values before date

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateTo={{fiware-dateTo}}&lastN={{fiware-lastN}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-lastN**: Entries number will be returned.	Method to retrieve the last "n" registered changes before a date.

Return:

```
The last "n" changes of the attribute before the specified date.
```

2.8.23 Retrieve raw values before date with pagination

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateTo={{fiware-dateTo}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number.	Method to retrieve all registered changes before a date.

Return:

```
From all registered changes of the attribute before the specified date will choose␣
↪the number of changes specified in "fiware-limit" and depending on the value of
"fiware-offset" will return certain changes.
```

2.8.24 Retrieve raw values before date with pagination and filetype

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?dateTo={{fiware-dateTo}}&hLimit={{fiware-limit}}&hOffset={{fiware-offset}}&filetype={{fiware-filetype}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-dateTo**: The final date and time until which the raw context information is desired. **fiware-limit**: Entries number for each page. **fiware-offset**: Entries offset number. **fiware-filetype**: File format will be saved all changes of the attribute. Currently, the only supported value and file type is “csv”.	Method to retrieve all registered changes before a date of an attribute.

Return:

```
Setting file with the number of changes of the attribute specified in “fiware-limit”␣
↪and from all changes registered before the specified date, certain changes are␣
↪chosen
depending on the value of “fiware-offset”.
```

2.8.25 Retrieve aggregated values

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?aggrMethod={{fiware-method}}&aggrPeriod={{fiware-period}}	GET	**url**: Link to the service will be consulted.	Method to retrieve aggregated value of an attribute based on "fiwaremethod" with a resolution defined in "fiware-period".
		port-sth: Port to connect with the service.	
		fiware-type: Type of the entity.	
		fiware-entity: ID of the entity which will be consulted in the service.	
		fiware-attr: ID of the attribute which will be consulted.	
		fiware-method: The aggregation method. The STH component supports the following aggregation methods: max (maximum value), min (minimum value), sum (sum of all the samples) and sum2 (sum of the square value of all the samples) for numeric attribute values and occur for attributes values of type string. Combining the information provided by these aggregated methods with the number of samples, it is possible to calculate probabilistic values such as the average value, the variance as well as the standard deviation.	
		fiware-period: Aggregation period or resolution. A fixed resolution determines the origin time format and the possible offsets. The STH component supports the following aggregation periods: month, day, hour, minute and second.	

Return:

```
If the type is String then returns the number of samples and for each value that has␣
↪been registered the number of times it appears in the samples.

If the type is numerical then returns the number of samples of the attribute and␣
↪depending on the value of "fiwaremethod" will calculated a value from all samples.
```

2.8.26 Retrieve aggregate values between two dates

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?aggrMethod={{fiware-method}}&aggrPeriod={{fiware-period}}&dateFrom={{fiware-dateFrom}}&dateTo={{fiware-dateTo}}	GET	**url**: Link to the service will be consulted.	Method to retrieve aggregated value of an attribute between two dates based on "fiwaremethod" with a resolution defined in "fiware-period".
		port-sth: Port to connect with the service.	
		fiware-type: Type of the entity.	
		fiware-entity: ID of the entity which will be consulted in the service.	
		fiware-attr: ID of the attribute which will be consulted.	
		fiware-method: The aggregation method. The STH component supports the following aggregation methods: max (maximum value), min (minimum value), sum (sum of all the samples) and sum2 (sum of the square value of all the samples) for numeric attribute values and occur for attributes values of type string. Combining the information provided by these aggregated methods with the number of samples, it is possible to calculate probabilistic values such as the average value, the variance as well as the standard deviation.	
		fiware-period: Aggregation period or resolution. A fixed resolution determines the origin time format and the possible offsets. The STH component supports the following aggregation periods: month, day, hour, minute and second.	

Return:

```
If the type is String then returns the number of samples and for each value that has␣
↪been registered the number of times it appears in the samples.

If the type is numerical then returns the number of samples of the attribute and␣
↪depending on the value of "fiwaremethod" will calculated a value from all samples.
```

2.8.27 Retrieve aggregate values after date

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?aggrMethod={{fiware-method}}&aggrPeriod={{fiware-period}}&dateFrom={{fiware-dateFrom}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-method**: The aggregation method. The STH component supports the following aggregation methods: max (maximum value), min (minimum value), sum (sum of all the samples) and sum2 (sum of the square value of all the samples) for numeric attribute values and occur for attributes values of type string. Combining the information provided by these aggregated methods with the number of samples, it is possible to calculate probabilistic values such as the average value, the variance as well as the standard deviation. **fiware-period**: Aggregation period or resolution. A fixed resolution determines the origin time format and the possible offsets. The STH component supports the following aggregation periods: month, day, hour, minute and second. **fiware-dateFrom**: The starting date and time from which the aggregated time series information is desired.	Method to retrieve aggregated value of an attribute between two dates based on "fiwaremethod" with a resolution defined in "fiware-period".

Return:

```
If the type is String then returns the number of samples and for each value that has␣
↪been registered the number of times it appears in the samples.

If the type is numerical then returns the number of samples of the attribute and␣
↪depending on the value of "fiwaremethod" will calculated a value from all samples.
```

2.8.28 Retrieve aggregate values before date

URL	Method	URL Params	Definition
http://{{url}}:{{port-sth}}/STH/v1/contextEntities/type/{{fiware-type}}/id/{{fiware-entity}}/attributes/{{fiware-attr}}?aggrMethod={{fiware-method}}&aggrPeriod={{fiware-period}}&dateTo={{fiware-dateTo}}	GET	**url**: Link to the service will be consulted. **port-sth**: Port to connect with the service. **fiware-type**: Type of the entity. **fiware-entity**: ID of the entity which will be consulted in the service. **fiware-attr**: ID of the attribute which will be consulted. **fiware-method**: The aggregation method. The STH component supports the following aggregation methods: max (maximum value), min (minimum value), sum (sum of all the samples) and sum2 (sum of the square value of all the samples) for numeric attribute values and occur for attributes values of type string. Combining the information provided by these aggregated methods with the number of samples, it is possible to calculate probabilistic values such as the average value, the variance as well as the standard deviation. **fiware-period**: Aggregation period or resolution. A fixed resolution determines the origin time format and the possible offsets. The STH component supports the following aggregation periods: month, day, hour, minute and second. **fiware-dateTo**:The final date and time until which the aggregated time series information is desired.	Method to retrieve aggregated value of an attribute after a date based on "fiwaremethod" with a resolution defined in "fiware-period".

Return:

```
If the type is String then returns the number of samples and for each value that has␣
↪been registered the number of times it appears in the samples.

If the type is numerical then returns the number of samples of the attribute and␣
↪depending on the value of "fiwaremethod" will calculated a value from all samples.
```

2.9 ANNEX 4: Cygnus RESTFul API

2.9.1 Retrieve admin login v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/login	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to retrieve the logging level of Cygnus.

Return:

```
In case of performing the method correctly, returns the logging level of cygnus and␣
↪"200 OK".

In case of wrong execution, return "500 Internal Server Error".
```

2.9.2 Update admin login v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/log?level={{fiware-logLevel}}	PUT	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-logLevel**: New level of log we want assigned. The valid loggins level are: *debug*, *info*, *warning*, *error* and *fatal*.	Method to update the logging level of Cygnus for an admin.

Return:

```
In case of performing the method correctly, returns "200 OK".

In case of wrong execution or invalid level, return "400 Bad Request".
```

2.9.3 Create agent parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}?param={{fiware-param}}&value={{fiware-paramValue}}	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to create. **fiware-paramValue**: Value of parameter we want to create.	Method to create a new parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, existing value or invalid␣
↪path to the agent configuration file, return "success: false".
```

2.9.4 Retrieve agent parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}&param={{fiware-param}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to retrieve.	Method to retrieve a defined parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true" with the␣
↪attribute and it value.

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the agent configuration file, return "success: false".
```

2.9.5 Retrieve agent parameters v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file.	Method to retrieve all parameters in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true" with the␣
↪attributes and its values.

In case of wrong execution by invalid configuration file or invalid path to the agent␣
↪configuration file, return "success: false".
```

2.9.6 Update agent parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}?param={{fiware-param}}&value={{fiware-paramValue}}	PUT	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to update. **fiware-paramValue**: Value of parameter wue wat to update.	Method to create a new parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the agent configuration file, return "success: false".
```

2.9.7 Delete agent parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}?param={{fiware-param}}	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to delete.	Method to delete a parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the attribute or␣
↪invalid path to the agent configuration file, return "success: false".
```

2.9.8 Create instance parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/instance/usr/cygnus/conf/{{fiware-instance}}?param={{fiware-param}}&value={{paramValue}}	POST	**url**: Link to the service that will be consulted.	Method to create a new parameter in an instance configuration file.
		port-cygnus: Port to connect with the service.	
		fiware-instance: Name of instance configuration file.	
		fiware-param: Name of parameter we want to delete.	
		fiware-paramValue: Value of parameter we want to create.	

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, existing value or invalid␣
↪path to the instance configuration file, return "success: false".
```

2.9.9 Retrieve instance parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/instance/usr/cygnus/conf/{{fiware-instance}}?param={{fiware-param}}	GET	**url**: Link to the service that will be consulted.	Method to retrieve a defined parameter in an instance configuration file.
		port-cygnus: Port to connect with the service.	
		fiware-instance: Name of instance configuration file.	
		fiware-param: Name of parameter we want to retrieve.	

Return:

```
In case of performing the method correctly, returns "success: true" with the␣
↪attribute and it value.

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the instance configuration file, return "success: false".
```

2.9.10 Retrieve instance parameter v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/instance/usr/cygnus/conf/{{fiware-instance}}?param={{fiware-param}}&value={{fiware-paramValue}}	PUT	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-instance**: Name of instance configuration file. **fiware-param**: Name of parameter we want to update. **fiware-paramValue**: Value of parameter we want to update.	Method to update a parameter in an instance configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the instance configuration file, return "success: false".
```

2.9.11 Delete parameter instance v0

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/admin/configuration/instance/usr/cygnus/conf/{{fiware-instance}}?param={{fiware-param}}	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-instance**: Name of instance configuration file. **fiware-param**: Name of parameter we want to delete.	Method to delete a parameter in an instance configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the attribute or␣
↪invalid path to the instance configuration file, return "success: false".
```

2.9.12 Create agent parameter v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}?param={{fiware-param}}&value={{fiware-paramValue}}	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to create. **fiware-paramValue**: Value of parameter we want to create.	Method to create a new parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, existing value or invalid␣
↪path to the agent configuration file, return "success: false".
```

2.9.13 Retrieve agent parameter v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}&param={{fiware-param}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to retrieve.	Method to retrieve a defined parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true" with the␣
↪attribute and it value.

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the agent configuration file, return "success: false".
```

2.9.14 Retrieve agent parameters v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file.	Method to retrieve all parameters in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true" with the␣
↪attribute and it value.

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the agent configuration file, return "success: false".
```

2.9.15 Update agent parameter v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}?param={{fiware-param}}&value={{fiware-paramValue}}	PUT	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to update. **fiware-paramValue**: Value of parameter we want to update.	Method to create a new parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the agent configuration file, return "success: false".
```

2.9.16 Delete agent parameter v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/configuration/agent/cygnus/apache-flume-1.4.0-bin/conf/{{fiware-agent}}?param={{fiware-param}}	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-agent**: Name of agent configuration file. **fiware-param**: Name of parameter we want to delete.	Method to delete a parameter in an agent configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the parameter or␣
↪invalid path to the instance configuration file, return "success: false".
```

2.9.17 Delete parameter instance v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/configuration/instance/usr/cygnus/conf/{{fiware-instance}}?param={{fiware-param}}	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-instance**: Name of instance configuration file. **fiware-param**: Name of parameter we want to delete.	Method to create a new parameter in an instance configuration file.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid configuration file, not found the attribute or␣
↪invalid path to the instance configuration file, return "success: false".
```

2.9.18 Retrieve version v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/version	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to retrieve the version of the running software.

Return:

```
In case of performing the method correctly, returns "success: true" and the version.

In case of wrong execution return "success: false".
```

2.9.19 Create grouping rules v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/groupingrules	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to create a new grouping rules, to join related data.

Body:

```
{
    "regex": "SmartSpot",
    "destination": "othersmartspots",
    "fiware_service_path": "smartspot",
    "fields": ["entityType"]
}
```

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution return "success: false".
```

2.9.20 Retrieve grouping rules v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/groupingrules	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to retrieve all grouping rules.

Body:

```
{
    "grouping_rules": [
        {
```

```
            "destination": "group_id",
            "fields": [
                "entityType"
            ],
            "fiware_service_path": "service_path",
            "id": 1,
            "regex": "entity_id"
        }
    ],
    "success": "true"
}
```

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution return "success: false".
```

2.9.21 Update grouping rules v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/groupingrules?id={{fiware-groupID}}	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-gropuID**: Name of group we want to update.	Method to update a grouping rules.

Body:

```
{
    "regex": "SmartSpot",
    "destination": "othersmartspots",
    "fiware_service_path": "smartspot",
    "fields": ["entityType"]
}
```

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution return "success: false".
```

2.9.22 Delete grouping rules v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/groupingrules?id={{fiware-groupID}}	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-gropuID**: Name of group we want to delete.	Method to delete a grouping rules defined as a parameter.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution return "success: false".
```

2.9.23 Create subscription for Orion v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/subscriptions?id=ngsi_version=1	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to create a subscription to one or many entities in Orion.

Body:

```
{
    "subscription":{
        "entities": [
            {
                "type": "entity_type",
                "isPattern": "false",
                "id": "entity_id"
            }
        ],
        "attributes": [],
        "reference": "http://reference_host:reference_port",
        "duration": "P1M",
        "notifyConditions": [
            {
                "type": "ONCHANGE",
                "condValues": []
            }
        ],
        "throttling": "PT5S"
    },
    "endpoint":{
        "host":"endpoint_host",
        "port":"endpoint_port",
        "ssl":"false",
        "xauthtoken":"234123123123123"
    }
}
```

- **isPattern**: Currently is hasn't use. Always 'false'.
- **id**: To which entity it wants to subscribe. In this case, to all entities of those type.
- **reference**: URL of client that it want to subscribe.
- **duration**: The duration of the subscription. In ISO 8601 standard format.
- **notifyConditions**: Define the launcher to notify the subscriptions. In this case, when change a value of attributes passed in the point "condValues" of an entity will launch a notification.
- **throttling**: Specify a minimum inter-notification arrival time. In this case, 5 seconds.
- **endpoint**: URL of Orion server.

Return:

```
In case of performing the method correctly, returns "success: true" with information␣
↪about the subscription (duration, throttling, subscriptionId).

In case of wrong execution return "success: false".
```

2.9.24 Create subscription for Orion v2 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/subscriptions?id=ngsi_version=2	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to create a subscription to one or many entities in Orion.

Body:

```
{
    "subscription":{
        "description": "subscription_description",
        "subject": {
            "entities": [
                {
                    "idPattern": ".*",
                    "type": "entity_type"
                }
            ],
            "condition": {
                "attrs": [
                    "attribute_entity"
                ],
                "expression": {
                    "q": "attribute_entity>40"
                }
            }
        },
        "notification": {
            "http": {
                "url": "http://localhost:1234"
            },
            "attrs": [
                "attribute_entity"
            ]
        },
        "expires": "2016-05-05T14:00:00.00Z",
        "throttling": 5
    },
    "endpoint":{
        "host":"<endpoint_host>",
        "port":"<endpoint_port>",
        "ssl":"false",
        "xauthtoken":"QsENv6AJj7blCqJ0YvfS5hMWYs"
    }
}
```

- **type**: To which entity it wants to subscribe. In this case, to all entities of those type.

- **condition**: Conditions that have to be fulfill for launch to notification.
- **expires**: The duration of the subscription, when it expires.
- **notification**: URL of client that it want to subscribe (cygnus_url) and the attributes that we want know.
- **throttling**: Specify a minimum inter-notification arrival time. In this case, 5 seconds.
- **endpoint: URL of Orion server.

Return:

```
In case of performing the method correctly, returns "success: true" with information␣
↪about the subscription (duration, throttling, subscriptionId).

In case of wrong execution return "success: false".
```

2.9.25 Retrieve subscription v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/subscriptions?id=ngsi_version={{orion-version}}&subscription_id={{fiware-subscription}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **orion-version**: Version of Orion to do this method.	Method to retrieve a subscription to one or many entities in Orion of cygnus.

Return:

```
In case of performing the method correctly, returns "success: true" with information␣
↪about the subscription.

In case of wrong execution by inexistent subscription id, invalid or not implemented␣
↪NGSI version or missing parameters return "success: false".
```

2.9.26 Retrieve subscriptions v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/subscriptions?id=ngsi_version={{orion-version}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **orion-version**: Version of Orion to do this method.	Method to retrieve all subscriptions in Orion of cygnus.

Return:

```
In case of performing the method correctly, returns "success: true" with information␣
↪about all subscriptions.

In case of wrong execution by invalid or not implemented NGSI version or missing␣
↪parameters return "success: false".
```

2.9.27 Delete subscription v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/subscriptions?id=ngsi_version}}&subscription_id={{subscription-id}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **orion-version**: Version of Orion to do this method. **subscription-id**: ID of subscription we want to delete.	Method to delete a subscription in Orion of cygnus.

Return:

```
In case of performing the method correctly, returns "success: true, result:␣
↪subscription deleted".

In case of wrong execution by invalid or not implemented NGSI version, wrong␣
↪subscription id, missing authentication token or missing fields return "success:␣
↪false".
```

2.9.28 Create appender v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{portcygnus}}/v1/admin/log?type=configuration&fiware-transient={{transient}}	POST	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to create an appender in a running logger.

Body:

```
{
    "appender": {
        "name":"appender_id",
        "class":"appender_class"
    },
    "pattern": {
        "layout":".....",
```

```
        "ConversionPattern":"....."
    }
}
```

Return:

```
In case of performing the method correctly, returns "success: true, result: Appender␣
↪'appender_id' put".

In case of wrong execution by invalid transient or without JSON return "success:␣
↪false".
```

2.9.29 Retrieve appender v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{portcygnus}}/v1/admin/log/appenders?name={{fiware-appender}}&transient={{fiware-transient}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file. **fiware-appender**: Name of appender we want to retrieve.	Method to retrieve a defined appender.

Return:

```
In case of performing the method correctly, returns "success: true" join to appender␣
↪information.

In case of wrong execution by invalid transient or invalid appender name return␣
↪"success: false".
```

2.9.30 Retrieve appenders v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/appenders?transient={{fiware-transient}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to retrieve all appenders in a running logger.

Return:

```
In case of performing the method correctly, returns "success: true" join to the␣
↪information about all appenders.

In case of wrong execution by invalid transient or appenders not be shown return␣
↪"success: false".
```

2.9.31 Update appender v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/appenders?transient={{fiware-transient}}	PUT	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to update an appender in a running logger.

Body:

```
{
    "appender": {
        "name":"appender_id",
        "class":"appender_class"
    },
    "pattern": {
        "layout":".....",
        "ConversionPattern":"....."
    }
}
```

Return:

```
In case of performing the method correctly, returns "{"success": "true","result":
↪"Appender 'appender_id' updated successfully"}".

In case of wrong execution by invalid transient or without JSON return "success:␣
↪false".
```

2.9.32 Delete appender v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/appenders?name={{fiware-appender}}&transient={{fiware-transient}}	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file. **fiware-appender**: Name of appender we want to delete.	Method to delete a defined appender.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid transient or invalid appender name return␣
↪"success: false".
```

2.9.33 Delete appenders v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/appenders?transient={{fiware-transient}}	DELETE	**url**: Link to the service that will be connected. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to delete all appenders.

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid transient return "success: false".
```

2.9.34 Create logger v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/loggers?transient={{fiware-transient}}	POST	**url**: Link to the service that will be connected. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to create a logger in a running Cygnus.

Body:

```
{
    "logger": {
        "name":"logger_id",
        "level":"....."
    }
}
```

Return:

```
In case of performing the method correctly, returns "success: true, result: logger␣
↪'logger_id' put."

In case of wrong execution by invalid transient or without JSON return "success:␣
↪false".
```

2.9.35 Retrieve logger v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/loggers?name={{fiware-logger}}&transient={{fiware-transient}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file. **fiware-logger**: Name of logger we want to retrieve.	Method to retrieve a defined logger.

Return:

```
In case of performing the method correctly, returns "success: true" join to logger␣
↪information.

In case of wrong execution by invalid transient or invalid logger name return␣
↪"success: false".
```

2.9.36 Retrieve loggers v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/loggers?transient={{fiware-transient}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to retrieve all loggers in a running Cygnus.

Return:

```
In case of performing the method correctly, returns "success: true" join to the␣
↪information about all loggers.

In case of wrong execution by invalid transient or loggers not be shown return␣
↪"success: false".
```

2.9.37 Update logger v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/loggers?transient={{fiware-transient}}	PUT	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-trasient**: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	Method to update a logger in a running Cygnus.

Body:

```
{
    "appender": {
        "name":"logger_id",
        "class":"logger_class"
    },
    "pattern": {
        "layout":".....",
        "ConversionPattern":"....."
    }
}
```

Return:

```
In case of performing the method correctly, returns "{"success": "true","result":
↪"Logger 'logger_id' updated successfully"}".

In case of wrong execution by invalid transient or without JSON return "success:␣
↪false".
```

2.9.38 Delete logger v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/loggers?name={{fiware-logger}}&transient={{fiware-transient}}	DELETE	**url**: Link to the service that will be consulted.	Method to delete a defined logger.
		port-cygnus: Port to connect with the service.	
		fiware-trasient: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	
		fiware-appender: Name of logger we want to delete.	

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid transient or invalid logger name return␣
↪"success: false".
```

2.9.39 Delete loggers v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/log/loggers?transient={{fiware-transient}}	DELETE	**url**: Link to the service that will be consulted.	Method to delete all loggers.
		port-cygnus: Port to connect with the service.	
		fiware-trasient: If false the appenders are retrieved from Cygnus, if true are retrieved from file.	

Return:

```
In case of performing the method correctly, returns "success: true".

In case of wrong execution by invalid transient return "success: false".
```

2.9.40 Metrics

The metrics defined in a Cygnus agent are:

- **incomingTransactions**: Number of incoming transactions. In other words, number of NGSI notifications received.
- **incomingTransactionRequestSize**: Total size of the requests related to incoming transactions (bytes).
- **incomingTransactionResponseSize**: Total size of the responses related to incoming transactions (bytes).
- **incomingTransactionError**: Number of incoming transactions causing an error.
- **serviceTime**: Average time between transaction requests reception and transaction responses sending.
- **outgoingTransactions**: Number of outgoing transactions. In other words, number of persistence operations.
- **outgoingTransactionRequestSize**: Total size of the requests related to outgoing transactions (bytes).
- **outgoingTransactionResponseSize**: Total size of the responses related to outgoing transactions (bytes).
- **outgoingTransactionError**: Number of outgoing transactions causing an error.

2.9.41 Retrieve metrics v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/metrics?reset={{fiware-reset}}	GET	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service. **fiware-reset**: If true return metrics and inmediatelly after they are deleted , if false metrics don't delete.	Method to retrieve all metrics for a Cygnus agent.

Return:

```
In case of performing the method correctly, returns "200 OK" join to all metrics␣
↪information.
```

2.9.42 Delete metrics v1 (beta)

URL	Method	URL Params	Definition
http://{{url}}:{{port-cygnus}}/v1/admin/metrics	DELETE	**url**: Link to the service that will be consulted. **port-cygnus**: Port to connect with the service.	Method to delete all metrics for a Cygnus agent (putting counters to zero).

Return:

```
In case of performing the method correctly, returns "200 OK".
```

2.10 ANNEX 5: IoT Agent LwM2M RESTFul API

2.10.1 Create device Provisioning

URL	Method	URL Params	Definition
http://{{url}}:{{port-iotagent}}/iot/devices	POST	**url**: Link to the service that will be consulted. **port-iotagent**: Port to connect with the service.	Method to create a device that we want to convert it in an entity.

Body:

```
{
    "devices": [
        {
            "device_id": "name_device",
            "entity_type": "type_id",
            "attributes": [
                {
                    "name": "attribute_name ",
                    "type": "attribute_type",
                }
            ],
            "lazy": [
                {
                    "name": "attribute_name ",
                    "type": "attribute_type"
                }
            ],
            "commands": [
                {
                    "name": "command_id",
                    "type": "command_type"
                }
            ],
            "internal_attributes": {
                "lmw2mResourceMapping": {
                    "attribute_id": {
                        "objectType":
                        "objecInstance":
                        "objectResource": 1
                    }
                }
            }
        }
    ]
}
```

- **lazy**: It a attribute to need explicit specification to can be read.
- **commands**: Commands you can send to the device.
- **internal_attributes**: Mapped of m2m language to entity attributes. Type, instance and source of each attribute of a entity.

Return:

```
In case of performing the method correctly, returns a confirmation that the device␣
↪has been created correctly "201 Created". In case of already any device with the␣
↪same name,
return "400 Bad Request" with this message:

{
        "name": "DUPLICATE_DEVICE_ID",
        "message": "A device with the same pair (Service, DeviceId) was found:..."
}
```

2.11 ANNEX 6: Deploy and use of Grafana

2.11.1 Creation and execution of the image in Docker

HOP Ubiquitous has a version of their fiware avaiable in a Docker container which "se puede" can download from Github. If you haven´t done this step we recommend you that you download the repository https://github.com/HOP-Ubiquitous/fiware-docker-infrastructure and you follow the recommend steps for its running.

Once we have execute the HOP Ubiquitous's container, we must have the next processes execute on Docker (we can see it using the command "docker ps" in terminal):

```
CONTAINER ID  IMAGE                            PORTS
3996f3dc01b5  iotagent                         0.0.0.0:5693->5683/udp dockercompose_
↪iotagent_1
e94378a74a38  smartsdk/quantumleap             0.0.0.0:8668->8668/tcp dockercompose_
↪quantumleap_1
a57a750c86c8  grafana/grafana                  0.0.0.0:3000->3000/tcp dockercompose_
↪grafana_1
e6ef4afd30f7  telefonicaiot/fiware-sth-comet   0.0.0.0:8666->8666/tcp dockercompose_
↪sth_1
fa5fbce0c89e  fiware/orion                     0.0.0.0:1026->1026/tcp dockercompose_
↪orion_1
6a22f61d772f  fiware/cygnus-ngsi 5050/tcp      0.0.0.0:8081->8081/tcp dockercompose_
↪cygnus_1
a29f3c7f682f  crate:1.0.5                      0.0.0.0:4200->4200/tcp, 0.0.0.0:4300->
↪4300/tcp, 5432-5532/tcp dockercompose_crate_1
a4652c49e3fa  mongo:3.2                        0.0.0.0:27016->27017/tcp dockercompose_
↪mongo_1
```

In case it is not executing some of this processes (as Grafana), we do the next process:

1. We stop the execution from Docker container – "**docker-compose down**".
2. We remove the image of the application which is not running. – "**docker rmi imageName**".
3. We back to execute the Docker container – "**docker-compose up - d**".

2.11.2 Data source definition to Grafana

Grafana is an open source API to generate dashboards from data contained in a CrateDB database. CrateDB is an open source SQL database design to store data of IoT devices. CrateDB is not suitable with Orion Context Broker, which manage the entire lifecycle of information, so we need a connector between CrateDB and Orion Context Broker.

QuantumLeap is an API to connect CrateDB and Orion Context Broker, in other words, it convert Orion Context Broker information in a suitable datatype with CrateDB and store it in this database.

2.11.3 Create QuantumLeap's subscription to Orion

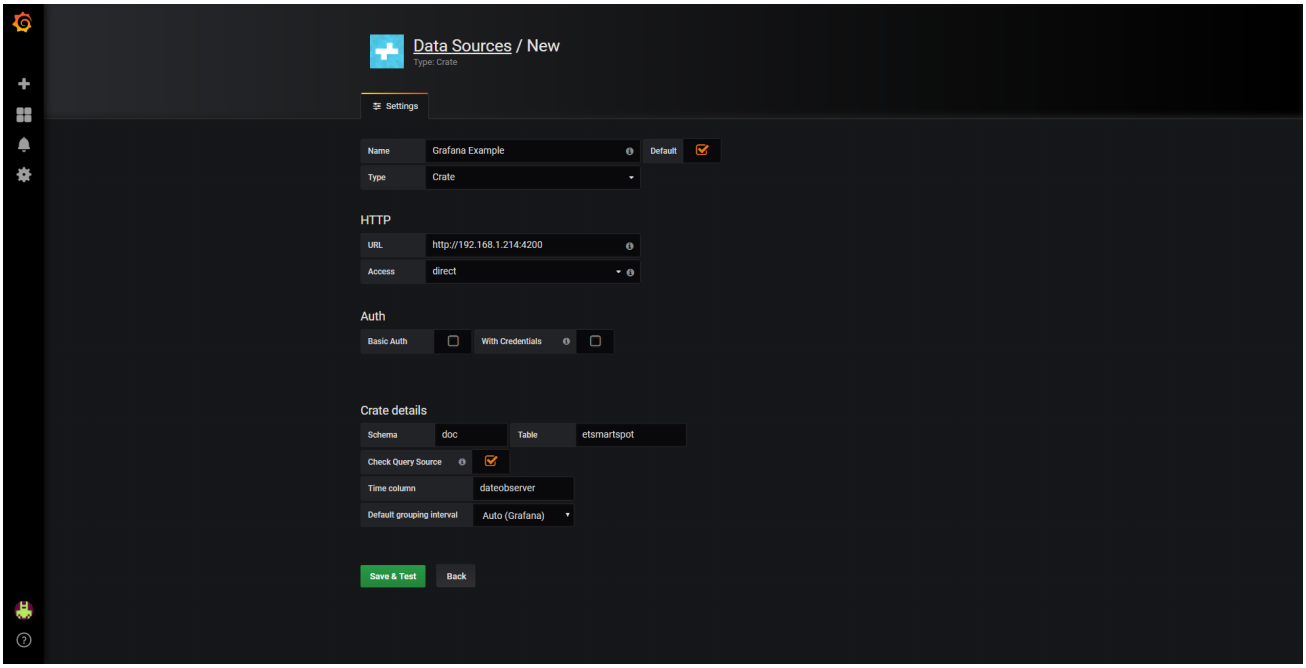
The next step is connect Data Source to Grafana. First we have to create a subscription from QuantumLeap to Orion to receive the changes from devices and it save on CrateDB database. For it, we execute the POSTMAN program and we open the Orion collection. We enter in the "**v2**" / "**subscriptions**" / "**create subscription v2**" section and the body section we defined the next subscription:

```
{
    "description": "grafana_subscription",
    "subject": {
        "entities": [
            {
                "idPattern": ".*",
                "type": "SmartSpot"
            }
        ]
    },
    "notification": {
        "http": {
            "url": "http://192.168.1.16:8668/notify"
        },
        "attrs": [
            "CO",
            "H2S",
            "NO2",
            "O3",
            "SO2",
            "batteryLevel",
            "humidity0",
            "humidity1",
            "temperature0",
            "temperature1",
            "nearDevicesHour",
            "nearDevicesMinute",
            "nearDevicesTenMinutes",
            "noise"
        ],
        "metadata": ["dateCreated", "dateModified"]
    }
}
```

We prove that the subscription has been made correctly and it is active. Once it is active, we check the QuantumLeap log to see if work correctly. To access the log, we execute the command "**docker logs –follow dockercompose_quantumleap_1**". We must prove that QuantumLeap receive the subscription and it support the attributes of an entity type that we want store in CrateDB.

We have already connect our devices with the CrateDB database from QuantumLeap. Now is the moment to defined our data source on Grafana. Before configured CrateDB, we can check if it is executing correctly. For it, we execute in Orion a GET method to the next address: http:/your_ip:4200/_sql and we check that exist the adequate data structure.

2.11.4 Connection of Grafana with CrateDB



The fi rst step is discover what our device ip is (with ifconfig command). Once we know our IP, we access to the next URL: http://your_ip:3000 (3000 is the port of grafana).

In the login from this URL we write in the user field: "**admin**" and in the password field: "**admin**". We enter the data source section and we leave the setting file like this.

- Data source type in our case is, as we have named before, CrateDB.
- The URL is you IP again, and 4200 is CrateDB port by default.
- When everything is working inside the localhost, the access is direct.
- QuantumLeap create a table "et" + name of the entity type which we have subscribed. In our case, being SmartSpot entities, the table it call "**etsmartspot**".
- The name of the index from time column is "dateobserved". This name can see in the CrateDB structure.

Grafana configuration to administrative level do it in the file **/etc/grafana/grafana.ini**

2.11.5 Create graphs in the dashboard

The first step is to go to create menu and then pulse in new dashboard and we select the option "**graph**" and will appears a new graph. We press in "**panel title**" and different options will appear.

The first that we have to do to create the graph is select the source where collect the data, for it in the "**Data Source**" section we have to press the "**default**" option and select "**Data Source**" that we want to use.

Once the source is selected we have to realise a query to the database so that Grafana creates us a graph with that data. In the next section the Grafana interface provide us a query structure to the database.

To select which attribute to show on the graph, we have to write on the "**value**" box of the "**Select**" section the attribute name which we want to look for. If we want to on the graph appears the change of more than one attribute we have to press on "+" symbol which place on the right part of the "**Select**" section and another "**Select**" section will be shown us.

The "**Select**" section has by default the "**Avg/Mean**" option. This option returns the data mean of that attribute. This option is useful if we want to know all changes of an attribute over time and there are multiple entities in the database. In the case that only we want to show the change of values of an attribute belonging to an entity, we have to insert in the "**Where**" section the comparison "**id == EntityName**" and we have to press on "**Avg/Mean**" and select the "**raw**" option so that return us the changes of the attribute underwent over time.

Once written the query we press the "**Query Inspector**" button and if the query which we have inserted it has been success then Grafana create a graph with the data of the attributes insert in the query, else show us an error and reason for the error.

2.11.6 Realise the query to the CrateDB database

To realise the query to the database we have to execute the "**POSTMAN**" program and inside "**fiware-grafana**" collection there is a method called "**retrieve grafana database**". This method returns all IDs of the attributes and for each entity the values of their attributes, for it in the method body we define the next query:

```
{
    "stmt": "SELECT * FROM \"doc\".\"etsmartspot\""
}
```

If we want to realise a query more specific to the database we only have to modify the method body and insert the query which we want to realise.

2.12 ANNEX 7: Perseo CEP

Perseo CEP is a distributed rule engine. It main objective is analyze the Orion's entities data and perform actions (events) to notify the clients about the service. To notify, Perseo can send a SMS or an e-mail. Events are received through a POST that contains the JSON representation, the rules to check and actions to generate in case the rules are complied with.

Basically, the Perseo operation is an event to trigger a rule, causing the selected values by EPL statements to send as a JSON to a set of URLs in a configuration file. In case of trigger many of a rule, each will generate their own POST to the defined URL.

2.13 ANNEX 7.1: Perseo-Core

Perseo is compose of Perseo-core and Perseo-fe. Perseo-core is the back-end of the CEP, her rule engine. It is responsible of the incoming event's checking through a POST that it contains the EPL rules defining and, in case of must to execute an action, it informs to the Perseo-fe through a POST request. The defined rules aren't stored persistently, but kept in memory, and it can be update from Perseo-fe.

The rules is defined from a request POST with the rule expression in EPL, rule language of Esper. Esper is a Java library that contains the rule engine and the events processing logic.

2.13.1 Retrieve rules

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/rules/	GET	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service.	Method to retrieve all exists rules.

Return:

```
In case of performing the method correctly, returns in the body the rules with a JSON␣
↪representation and "200 OK".

In case of not have any rule, return "[]".
```

2.13.2 Retrieve rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/rules/{{fiware-rule}}	GET	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service. **fiware-rule**: Name of the rule we want retrieve.	Method to retrieve the defined rule as parameter.

Return:

```
In case of performing the method correctly, returns in the body the defined rule with␣
↪a JSON representation and "200 OK".

In case of not have any rule with this name, return "404".
```

2.13.3 Create/Update rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/rules/	POST	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service.	Method to create a new rule or update it if exist the defined rule.

Body:

```
{
    "name": "
    CO_email",
    "text": "select *,\"CO_email\" as ruleName, *,smartspot.CO? as CO, smartspot.id?␣
↪as SmartSpot from pattern [every smartspot=iotEvent(cast(cast(SmartSpot?,String),
↪float)>1.5 and type=\"SmartSpot\")]",
    "action": [
        {
```

```
            "type": "email",
            "template": "CO ${SmartSpot} tiene un CO ${CO} (GEN RULE)",
            "parameters": {
                    "to": "user@hopu.eu",
                    "from": "admin@hopu.eu",
                    "subject": "${SmartSpot} ha cambiado"
            }
        }
    ]
}
```

Return:

```
In case of performing the method correctly, returns in the body the new rule created␣
↪or updated in a JSON representation and "200 OK".

In case of wrong execution, return "400 Bad Request".
```

2.13.4 Update rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/rules/	PUT	**url**: Link to the service that will be consulted.	Method to update a rule or set of rules.
		port-perseo-core: Port to connect with the service.	

Body:

```
{
    "name": "co_update",
    "text": "select *,\"co_update\" as ruleName from pattern [every␣
↪ev=iotEvent(cast(cast(CO?,String),float)>1.5 and type=\"SmartSpot\")]",
    "action": {
        "type": "update",
        "parameters": {
            "attributes": [
                {
                    "name": "abnormal",
                    "value": "true",
                    "type": "boolean"
                },
                {
                    "name": "CO",
                    "value": 12.34,
                    "type": "Number"
                }
            ]
        }
    }
}
```

Return:

```
In case of performing the method correctly, returns "200 OK" with an empty JSON␣
↪object "[]".

In case of wrong execution, return "400 Bad Request".
```

2.13.5 Delete rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/rules/{{fiware-rule}}	DELETE	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service. **fiware-rule**: Name of the rule we want retrieve.	Method to delete a defined rule as parameter.

Return:

```
In case of performing the method correctly, returns in the body the deleted rule in a␣
↪JSON representation and "200 OK".

In case of not have any rule with this name, return "200" OK with an empty JSON␣
↪object "[]".
```

2.13.6 Create event

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/events	POST	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service.	Method to send an event to the rule engine.

Body:

```
{
    "noticeId":"21f12c40-1ca6-11e4-a992-f1e158aa3052",
    "received":"2014-08-05T13:40:58.756Z",
    "id":"Prueba_de_ubicacion",
    "type":"SmartSpot",
    "isPattern":"false",
    "CO":"2.3202312",
    "CO type":"number",
    "TimeInstant":"2014-04-29T13:18:05Z",
    "TimeInstant type":"urn:x-ogc:def:trs:IDAS:1.0:ISO8601"
}
```

Return:

```
In case of performing the method correctly, returns "200 OK" with an empty JSON␣
↪object "[]".

In case of wrong execution, return "400 Bad Request".
```

2.13.7 Retrieve log

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseo-core}}/perseo-core/admin/log	GET	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service.	Method to retrieve the logging level of Perseo-Core.

Return:

```
In case of performing the method correctly, returns the logging level of Perseo-Core
↪and "200 OK". The loggins level can be "DEBUG", "INFO", "WARN", "WARNING",
"ERROR" and "FATAL".

In case of wrong execution, return "400 Bad Request".
```

2.13.8 Update log level

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseocore}}/perseo-core/admin/log?level={{perseo-level}}	PUT	**url**: Link to the service that will be consulted. **port-perseo-core**: Port to connect with the service. **perseo-level**: Logging level of Perseo-Core. Can be "*DEBUG*", "*INFO*", "*WARN*", "*WARNING*", "*ERROR*" and "*FATAL*".	Method to update the logging level of Perseo-Core.

Return:

```
In case of performing the method correctly, returns "200 OK" with an empty JSON
↪object "[]".

In case of wrong execution by incorrect value in "perseo-level", return "400 Bad
↪Request".
```

2.14 ANNEX 7.2: Perseo-Fe

Perseo-fe is a component that together with Perseo-core form the Perseo fiware. Perseo-fe refresh all of set of rules of Perseo-core. When Perseo-core send a "action" to Perseo-fe, it is responsible of send an action via SMS, email o HTTP.

In order for Perseo can send a notification, it must have configured the following servers: SMPP, SMTP and HTTP.

SMPP server allows Perseo to send an action to a recipient via SMS. In order to create the SMS, the action must have in one of its fields the number of recipient who that to will receive the SMS.

SMTP server allows Perseo to send to a recipient an action via SMS. In order to create the e-mail, the action must have in one of its field the e-mail of the sender and the e-mail of the recipient.

HTTP server allows an HTTP POST to be performed to a URL provided in a field of the action.

2.14.1 Retrieve version

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/version	GET	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service.	Method to retrieve the currently version of Perseo.

Return:

```
In case of performing the method correctly, returns "200 OK" with information of␣
↪Perseo version.

In case of wrong execution, return "400 Bad Request".
```

2.14.2 Retrieve rules

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/rules	GET	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service.	Method to retrieve all rules from Perseo.

Return:

```
In case of performing the method correctly, returns in the body the rules with a JSON␣
↪representation and "200 OK".

In case of not have any rule, return "[]".
```

2.14.3 Retrieve rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/rules/{{rule-id}}	GET	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service. **rule-id**: ID of the rule which will be retrieved.	Retrieve a rule from the service.

Return:

```
In case of performing the method correctly, returns in the body the defined rule with␣
↪a JSON representation and "200 OK".

In case of not have any rule with this name, return "404".
```

2.14.4 Delete rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/rules/{{rule-id}}	DELETE	**url**: Link to the service that will be consulted.	Method to delete a rule.
		port-perseoFe: Port to connect with the service.	
		rule-id: ID of the rule which will be deleted from Perseo.	

Return:

```
In case of performing the method correctly, returns in the body the deleted rule in a␣
↪JSON representation and "200 OK".

In case of not have any rule with this name, return "200" OK with an empty JSON␣
↪object "[]".
```

2.14.5 Create rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/rules	POST	**url**: Link to the service that will be consulted.	Method to create a new rule.
		port-perseoFe: Port to connect with the service.	

Body:

```
{
    "name": "CO_email",
    "text": "select *,\"CO_email\" as ruleName, *,ev.CO? as CO, ev.id? as SmartSpot␣
↪from pattern [every ev=iotEvent(cast(cast(CO?,String),float)>1.5 and␣
↪type=\"SmartSpot\")]",
    "action": [
        {
            "type": "email",
            "template": "SmartSpot ${SmartSpot} has CO ${CO} (GEN RULE)",
            "parameters": {
                "to": "user@hopu.eu",
                "from": "admin@hopu.eu",
                "subject": "${CO} has changed"
            }
        }
    ]
}
```

Return:

```
In case of performing the method correctly, returns in the body the new rule created␣
↪in a JSON representation and "200 OK".

In case of wrong execution, return "400 Bad Request".
```

2.14.6 Create notification

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/notices	POST	**url**: Link to the service that will be consulted.	Method to create a new notification.
		port-perseoFe: Port to connect with the service.	

Body:

```
{
    "contextResponses": [
        {
            "contextElement":{
                "attributes": [
                    {
                        "metadatas": [
                            {
                                "name": "previousValue",
                                "type": "Number",
                                "value": "11"
                            },
                            {
                                "name": "actionType",
                                "type": "Text",
                                "value": "update"
                            }
                        ],
                        "name": "A",
                        "type": "Number",
                        "value": "12"
                    }
                ],
                "id": "Prueba_de_ubicacion",
                "isPattern": "false",
                "type": "SmartSpot"
            },
            "statusCode": {
                "code": "200",
                "reasonPhrase": "OK"
            }
        }
    ],
    "originator": "localhost",
    "subscriptionId": "123456789"
}
```

2.14.7 Retrieve visual rules

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/m2m/vrules	GET	**url**: Link to the service that will be consulted.	Method to retrieve all visual rules from Perseo.
		port-perseoFe: Port to connect with the service.	

Return:

```
All visual rules from Perseo.
```

2.14.8 Retrieve visual rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/m2m/vrules/{{rule-id}}	GET	**url**: Link to the service that will be consulted.	Retrieve a visual rule from the service.
		port-perseoFe: Port to connect with the service.	
		rule-id: ID of the visual rule which will be retrieved.	

Return:

```
In case of performing the method correctly, returns in the body the defined visual␣
↪rule with a JSON representation and "200 OK".

In case of not have any rule with this name, return "404".
```

2.14.9 Create visual rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/m2m/vrules	POST	**url**: Link to the service that will be consulted.	Method to create a new visual rule.
		port-perseoFe: Port to connect with the service.	

Body:

```
{
    "name" : "prueba-test",
    "active" : 1,
    "cards" : [
        {
            "type" : "ActionCard",
            "actionData" : {
                "userParams" : [
                    {
                        "name" : "mail.from",
                        "value" : "dca_support@tid.es"
                    }
                ],
                "name" : "email",
                "type" : "SendEmailAction"
            },
            "id" : "card_42",
            "connectedTo" : [ ]
        }
    ]
}
```

Return:

```
In case of performing the method correctly, returns in the body the new visual rule␣
↪created in a JSON representation and "200 OK".

In case of wrong execution, return "400 Bad Request".
```

2.14.10 Delete visual rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/m2m/vrules/{{vrule-id}}	DELETE	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service. **vrule-id**: ID of the virtual rule which will be deleted from Perseo.	Method to delete a rule.

Return:

```
In case of performing the method correctly, returns in the body the deleted virtual␣
↪rule in a JSON representation and "200 OK".

In case of not have any virtual rule with this name, return "200" OK with an empty␣
↪JSON object.
```

2.14.11 Update visual rule

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/m2m/vrules/{{vrule-id}}	PUT	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service. **vrule-id**: ID of the virtual rule which will be updated from Perseo.	Method to update an existing visual rule.

Body:

```
{
    "name" : "prueba-test",
    "active" : 1,
    "cards" : [
        {
            "type" : "ActionCard",
            "actionData" : {
                "userParams" : [
                    {
                        "name" : "mail.from",
                        "value" : "dca_support@tid.es"
                    }
                ],
                "name" : "email",
                "type" : "SendEmailAction"
```

```
            },
            "id" : "card_42",
            "connectedTo" : [ ]
        }
    ]
}
```

Return:

```
In case of performing the method correctly, returns in the body the visual rule␣
↪updated in a JSON representation and "200 OK".

In case of wrong execution, return "400 Bad Request".
```

2.14.12 Retrieve log

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/admin/log	GET	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service.	Method to retrieve the logging level of Perseo-Fe.

Return:

```
In case of performing the method correctly, returns the logging level of Perseo-Core␣
↪and "200 OK". The loggins level can be "DEBUG", "INFO", "WARN", "WARNING",
"ERROR" and "FATAL".

In case of wrong execution, return "400 Bad Request".
```

2.14.13 Retrieve log level

URL	Method	URL Params	Definition
http://{{url}}:{{port-perseoFe}}/admin/log?level={{perseo-level}}	PUT	**url**: Link to the service that will be consulted. **port-perseoFe**: Port to connect with the service. **perseo-level**: Logging level of Perseo-Core. Can be "*DEBUG*", "*INFO*", "*WARN*", "*WARNING*", "*ERROR*" and "*FATAL*".	Method to update the logging level of Perseo-Fe.

Return:

```
In case of performing the method correctly, returns "200 OK" with an empty JSON␣
↪object "[]".

In case of wrong execution by incorrect value in "perseo-level", return "400 Bad␣
↪Request".
```

2.15 ANNEX 7.3: Perseo Use Cases

As we have seen in the introduction, Perseo-Fe makes an action before changes of values of Entities of Orion. These actions usually are notifications to user. Currently, the notification to user can be done of the following ways:

- Update. Update the attributes of an Orion entity when some value of an attribute is different to expected.
- Post. Send to a REST service a HTTP message to notify different situation than normal.
- Twitter. Post a tweet with the information relevant to notify some value.
- E-mail. Send an e-mail when an entity has some attribute a value different to the optimum.
- SMS. Send a SMS when the values of an Orion entity are different to expected.

In addition to normal notifications, there are complex notifications which add on the notification message statistics about the values (as means, totals, etc.).

For our use cases, we go to notify to user when CO value in a concrete SmartSport exceed a determinate value.

In first place, we must make a series of initial steps for the connection of Orion and Perseo:

1. Create the subscription from Perseo-Fe to Orion.

Perseo-Fe must be subscribe to Orion for be notified when the values of one or several attributes of one or several entities of a type concrete change. In our case, we want to be notify when the CO value of the SmartSpot change. We make the next subscription on Orion:

```
{
    "entities": [
        {
            "type": "SmartSpot",
            "isPattern": "true",
            "id": ".*"
        }
    ],
    "attributes": [
        "CO"
    ],
    "reference": "http://perseo-fe:9090/notices",
    "duration": "P1Y",
    "notifyConditions": [
        {
            "type": "ONCHANGE",
            "condValues": [
                "CO"
            ]
        }
    ],
    "throttling": "PT1S"
}
```

2. Create a SmartSpot.

In case don´t have some SmartSpot create on Orion, we create the next test SmartSpot.

```
{
    "contextElements": [
        {
            "type": "SmartSpot",
            "isPattern": "false",
```

```
            "id": "Prueba_de_ubicacion",
            "attributes": [
                {
                    "name": "CO",
                    "type": "Number",
                    "value": "0.7626428598980013"
                }
            ]
        }
    ],
    "updateAction": "APPEND"
}
```

We can verify if Perseo-Fe receive correct notifications from Orion checking their log, for it, we execute about terminal the next command: **docker logs dockercompose_perseo-fe_1 –follow**

2.15.1 Update

It serves to update or put attributes of an Orion entity when the value of the CO of the SmartSport excedes 1'5. The steps to follow:

1. Create an UPDATE rule in Peseo-Fe.

We create the next rule which add the attributes "abnormal" and "other" in case the CO of any "SmartSpot" entity have a value greater than 1'5.

```
{
    "name": "CO_update",
    "text": "select *,\"CO_update\" as ruleName from pattern [every␣
↪ev=iotEvent(cast(cast(CO?,String),float)>1.5 and type=\"SmartSpot\")]",
    "action": {
        "type": "update",
        "parameters": {
            "attributes": [
                {
                    "name": "abnormal",
                    "value": "true",
                    "type": "boolean"
                },
                {
                    "name": "other",
                    "value": 12.34,
                    "type": "Number"
                }
            ]
        }
    }
}
```

2. Update our SmartSpot.

We update the value of the CO of our SmartSpot of test increasing the CO to a value greater than 1'5.

```
{
    "contextElements": [
        {
            "type": "SmartSpot",
```

```
            "isPattern": "false",
            "id": "Prueba_de_ubicacion",
            "attributes": [
                {
                    "name": "CO",
                    "type": "Number",
                    "value": "1.7626428598980013"
                }
            ]
        }
    ],
    "updateAction": "UPDATE"
}
```

3. Check the results.

We back obtain the entity of test of our SmartSpot to check that , in effect, these two attributes have been created.

```
[
    {
        "id": "Prueba_de_ubicacion",
        "type": "SmartSpot",
        "CO": {
            "type": "Number",
            "value": "1.7626428598980013",
            "metadata": {}
        },
        "abnormal": {
            "type": "boolean",
            "value": "true",
            "metadata": {}
        },
        "other": {
            "type": "Number",
            "value": "12.34",
            "metadata": {}
        }
    }
]
```

2.15.2 Post

In this example: Perseo-Fe sends a HTTP message when the value of the CO of the SmartSpot of test exceed 10'5. The steps to follow:

1. Create a POST rule in Perseo-Fe.

When the CO of some SmartSpot entity is greater than 10'5, we send a HTTP message with PUT method to URL http://localhost:8080/${type}/${id}. We define also header and body of the method.

```
{
    "name":"CO_post",
    "text":"select \"CO_post\" as ruleName, * from pattern␣
↪[everyev=iotEvent(cast(cast(CO?,String),float)>10.5 and type=\"SmartSpot\")]",
    "action":{
        "type":"post",
        "parameters":{
```

```
            "url": "http://localhost:8080/${type}/${id}",
            "method": "PUT",
            "headers": {
                "Content-type": "application/json",
                "X-CO": "${CO}"
            },
            "qs": {
                "bp": "${CO}",
                "id": "${id}",
                "${type}": "${type}"
            },
            "json": {
                "CO": "${CO}"
            }
        }
    }
}
```

2. Update our SmartSpot.

We update the value of the CO of our SmartSpot of test increasing the CO to a value greater that 10‘5.

```
{
    "contextElements": [
        {
            "type": "SmartSpot",
            "isPattern": "false",
            "id": "Prueba_de_ubicacion",
            "attributes": [
                {
                    "name": "CO",
                    "type": "Number",
                    "value": "20.7626428598980013"
                }
            ]
        }
    ],
    "updateAction": "UPDATE"
}
```

2.15.3 Twitter

In this use case, Perseo-Fe publish a tweet in the Twitter account that we have passed the keys when, again, the CO value of the SmartSpot exceeds 1’5.

To normal operation of this action, we must obtain the Twitter keys to Perseo can connect with a defined Twitter account.

1. Create a Twitter rule in Perseo-Fe.

This rule is launched when the CO value of any entity of SmartSpot type exceeds 1’5. In this moment, Perseo publish the next Tweet: “SmartSpot ${SmartSpot} tiene un CO ${CO}. Prueba realizada en @HOPUbiquitous”. As parameters, we must have passed the Twitter keys.

```
{
    "name":"CO_twitter",
    "text":"select *,\"CO_twitter\" as ruleName, *, ev.CO? as CO, ev.id? as SmartSpot␣
↪from pattern [every ev=iotEvent(cast(cast(CO?,String),float)>1.5 and␣
↪type=\"SmartSpot\")]",
```

```
    "action":{
        "type":"twitter",
        "template":"SmartSpot ${SmartSpot} tiene un CO ${CO}. Prueba realizada en␣
↪@HOPUbiquitous",
        "parameters":{
            "consumer_key": "XX",
            "consumer_secret": "XX",
            "access_token_key": "XX",
            "access_token_secret": "XX"
        }
    }
}
```

2. Update our SmartSpot.

We update the CO value of our SmartSpot increase the CO to a value greater than 1'5.

```
{
    "contextElements": [
        {
            "type": "SmartSpot",
            "isPattern": "false",
            "id": "Prueba_de_ubicacion",
            "attributes": [
                {
                    "name": "CO",
                    "type": "Number",
                    "value": "1.7626428598980013"
                }
            ]
        }
    ],
    "updateAction": "Update"
}
```

2.15.4 E-mail

In this example, Perseo-Fe send an e-mail to defined recipient when, again, the CO value of the SmartSpot exceeds 1'5.

To normal operation of this action, first, we must configured Perseo-Fe to define the SMTP server parameters.

1. Configure the SMTP Server.

We must configure SMTP server in the Perseo-Fe configuration. To that, in docker-compose.yml file, inside of Docker container fiware-docker-infraestructure of Hop Ubiquitous, we added in the Perseo-Fe configuration file the next configuration lines:

- PERSEO_SMTP_HOST = smtp.servidor.com -> Host of the SMTP server.
- PERSEO_SMTP_PORT = 465 -> Port of the SMTP server.
- PERSEO_SMTP_SECURE = true -> true if SSL should be used with the SMTP server.
- PERSEO_SMTP_AUTH_USER = usuario@hopu.eu -> Authentication data, the username.
- PERSEO_SMTP_AUTH_PATH = xxx -> Authentication data, the password for the user.

In order for this changes have effect, we must reset the Docker container in case it is already booted.

2. Create an E-mail rule in Perseo-Fe.

This rule is launched when the CO value of any entity of SmartSpot type exceeds 1'5. In this moment, Perseo send the next e-mail: "CO ${SmartSpot} tiene un CO ${CO}. Es peligroso.". The e-mail will have "${SmartSpot} ha cambiado" as subject, the sender will be usuario@hopu.eu and the recipient will be usuario@gmail.com.

```
{
    "name": "CO_email",
    "text": "select *,\"CO_email\" as ruleName, *,smartspot.CO? as CO, smartspot.id?␣
↪as Smart-Spot from pattern [every smartspot=iotEvent(cast(cast(SmartSpot?,String),
↪float)>1.5 and type=\"SmartSpot\")]",
    "action": [
        {
            "type": "email",
            "template": "CO ${SmartSpot} tiene un CO ${CO}. Es peligroso.",
            "parameters": {
                "to": "usuario@gmail.com",
                "from": "usuario@hopu.eu",
                "subject": "${SmartSpot} ha cambiado"
            }
        }
    ]
}
```

3. Update our SmartSpot.

We update the CO value of our SmartSpot increase the CO to a value greater than 1'5.

```
{
    "contextElements": [
        {
            "type": "SmartSpot",
            "isPattern": "false",
            "id": "Prueba_de_ubicacion",
            "attributes": [
                {
                    "name": "CO",
                    "type": "Number",
                    "value": "20.7626428598980013"
                }
            ]
        }
    ],
    "updateAction": "Update"
}
```

2.15.5 SMS

In this example, Perseo-Fe send an SMS to defined recipient when, again, the CO value of the SmartSpot exceeds 1'5.

To normal operation of this action, first, we must configured Perseo-Fe to define the SMPP server parameters and the SMS parameters.

1. Configure the SMPP Server.

We must configure SMTP server in the Perseo-Fe configuration. To that, in docker-compose.yml file, inside of Docker container fiware-docker-infraestructure of Hop Ubiquitous, we added in the Perseo-Fe configuration file the next configuration lines:

- PERSEO_SMS_URL = localhost -> URL for sending SMSs (SMPP Adapter).
- PERSEO_SMS_API_KEY API KEY = XXX -> API KEY for sending SMS, if necessary.
- PERSEO_SMS_API_SECRET API SECRET = XXX -> API SECRET for sending SMS, if necessary.
- PERSEO_SMS_FROM = 987654123 -> Field from for the outgoing SMSs.
- PERSEO_SMPP_HOST = localhost -> Host of the SMPP server.
- PERSEO_SMPP_PORT = 3550 -> Port of the SMPP server.
- PERSEO_SMPP_SYSTEMID = user -> SystemID for the user of the SMPP server.
- PERSEO_SMPP_PASSWORD = password -> Password for the user of the SMPP server.
- PERSEO_SMPP_FROM = 987654123 -> Number from SMS are sending by SMPP server.
- PERSEO_SMPP_ENABLED = true -> SMPP is default method for SMS instead of use SMS gateway.

In order for this changes have effect, we must reset the Docker container in case it is already booted.

2. Create a SMS rule in Perseo-Fe.

This rule is launched when the CO value of any entity of SmartSpot type exceeds 1'5. In this moment, Perseo send the next SMS: "SmartSpot ${SmartSpot} tiene un CO ${CO}.". The recipient will be 636254XXX.

```
{
    "name":"CO_sms",
    "text":"select *,\"CO_sms\" as ruleName, *, smartspot.CO? as CO, smartspot.id? as␣
↪SmartSpot from pattern [every smartspot=iotEvent(cast(cast(CO?,String),float)>1.5␣
↪and type=\"SmartSpot\")]",
    "action":{
        "type":"sms",
        "template":"SmartSpot ${SmartSpot} tiene un CO ${CO}.",
        "parameters":{
            "to":"636254XXX"
        }
    }
}
```

3. Update our SmartSpot.

We update the CO value of our SmartSpot increase the CO to a value greater than 1'5.

```
{
    "contextElements": [
        {
            "type": "SmartSpot",
            "isPattern": "false",
            "id": "Prueba_de_ubicacion",
            "attributes": [
                {
                    "name": "CO",
                    "type": "Number",
                    "value": "20.7626428598980013"
                }
            ]
        }
    ],
    "updateAction": "Update"
}
```

CHAPTER 3

HOP Ubiquitous - Relevant Links

- HOP Ubiquitous Web: http://www.hopu.eu/
- HOP Ubiquitous Smartcities: http://smartcities.hopu.eu/
- HOP Ubiquitous Dashboard: https://live.hopu.eu
- Homard App: http://staging.hopu.eu
- Homard Wiki: https://homard.hopu.eu/indexPage/wiki.html