yooBee Software Development Kit, v2.0

Blooloc

July 26, 2017

Contents

Introduction **2**
Scope . . . 2
Intended Audience . . . 2
History . . . 2

1 Overview **4**

2 yooBee Positioning API **6**
Data model . . . 6
Protocol . . . 7
Transport . . . 7
Versioning . . . 7
Authentication . . . 7
GET requests: a Practical Example . . . 8
PUT request: a Practical Example . . . 9
Infrastructure API . . . 10
Venues . . . 10
Floors . . . 15
Beacons . . . 17
Mobiles . . . 19
Positioning API . . . 20
Querying . . . 20
Streaming . . . 22
Event API . . . 24
Button Press . . . 25
Status API . . . 26
Beacons . . . 26
Basestations . . . 27
Tags . . . 28

Introduction

Scope

This document defines the SDK and API available for the integration of the yooBee technology into different applications.

PLEASE NOTE! The functionality of the SDK and the APIs are still evolving and thus subject to change.

Intended Audience

This document is intended for developers who are building an application that interfaces with the yooBee platform, be it on the mobile phone (or tablet) or from within the back-office.

History

Revision	Date	Author(s)	Description
0.1	May 13 2014	KD	Created
0.2	July 2 2014	KD	Updated to reflect authentication and expanded SDK information
0.3	July 14 2014	KD	Added preliminary note
0.4	December 16 2014	BD	Updated Database model and API results
0.4.1	January 28 2015	BD	Updated API working functionalities
0.4.2	April 9 2015	WD	Naming modifications
0.4.3	July 1 2015	BD	Updated API Authentication
0.6	July 20 2015	KD	Cloud SDK Changes
0.7	Aug 14 2015	KD	Updated URL for cloud API endpoint
1.0	Nov 17 2015	KD	Updated for 1.0 release
1.1	Mar 25 2016	KD	Described new features (walking, state)
1.2	Apr 22 2016	KD	Remove deprecated zone-based APIs
1.3	Nov 15 2016	YG	Described new API calls (floors, events)

Revision	Date	Author(s)	Description
1.4	Dec 6 2016	TD	Added request and response examples of several calls.
1.5	Dec 20 2016	TD	Converted to Markdown (Pandoc). Added PUT request info.
1.6	Jan 11 2017	TD	Added global-transform and lat-lng. Added description to Status.
1.7	Feb 28 2017	TD	Added mobile movement-type to infrastructure call. Added venue-id to tag status (filter & output).
2.0	May 19 2017	TD	Replaced API v1.0 with v2.0

1

Overview

Figure 1.1 situates the two APIs that are described in this document:

The yooBee Positioning API is a web API which allows applications and customer IT systems to query and interact with positioning data.

The yooBee Mobile SDK is an SDK (for Android and iOS) which allows the development of location-aware apps. The yooBee Mobile SDK is also a consumer of the yooBee Positioning API.

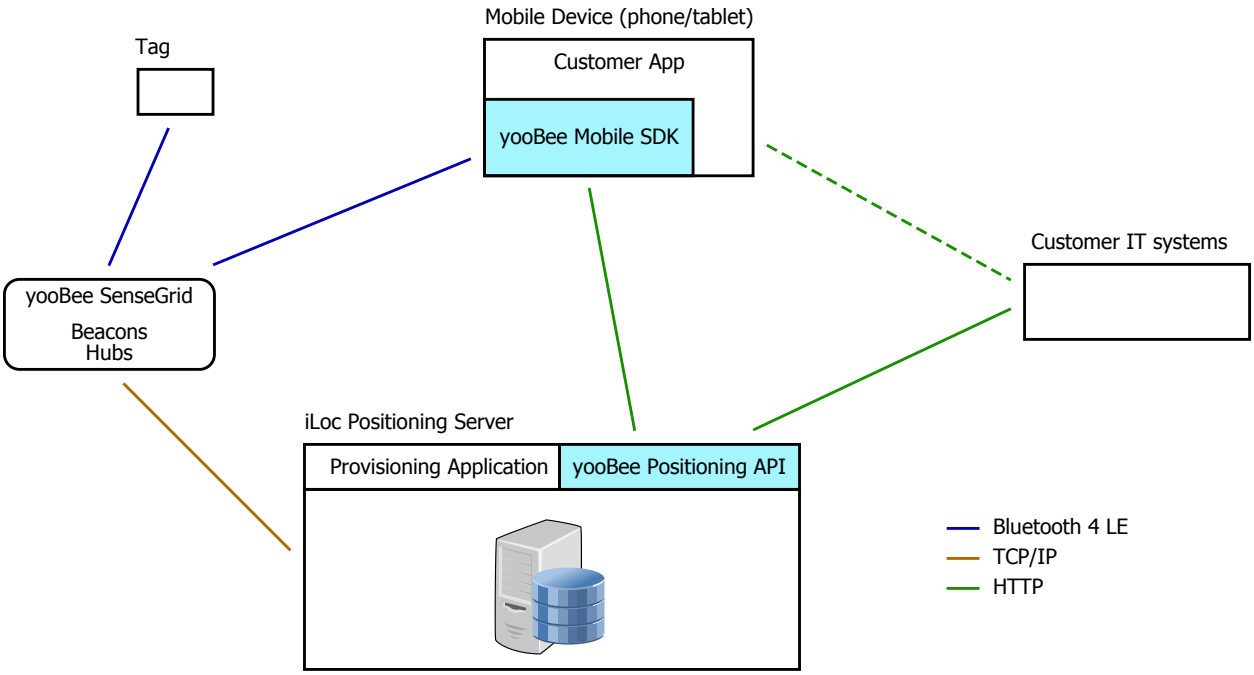

Figure 1.1: Situating the Positioning API and Mobile SDK

A yooBee system can be rolled out in two ways:

Cloud deployment : the software is hosted in the cloud on BlooLoc's servers, and is available as a service. Beacons and mobile tags communicate (using BLE) with basestations, which themselves connect to the cloud using a secure communication. Phones and tablets interact with the cloud system for positioning, and integration happens by interacting with web services provided in the cloud. This is for example the case if you bought a Starter Kit.

Local deployment : the software is hosted locally (e.g. within a venue, in a private cloud, or in a portable system). This is also the case if you bought a Developer Kit, which combines the function of a basestation and a local server.

In terms of APIs and methods to interact with the positioning system, there is little difference between the two deployment systems.

2

yooBee Positioning API

This API defines methods to query the position of a mobile node, or to track the position of a node over-time.

Data model

Affiliation user affiliation/organization, which may act as the owner of one or more venues or one or more mobiles

User affiliated with an affiliation and its own access credentials

Venue a venue is a roll-out of yooBee equipment in a building. The roll-out may cover part of a floor, a whole floor, or multiple floors. The definition of a venue may also define a transformation matrix that maps local (X/Y) coordinates (in meters) to Lat/Long coordinates (WGS84). If a venue includes multiple buildings that are interconnected by indoor or outdoor passages that also provide localization, then a venue may also be a group of such (interconnected) buildings.

Floor defines a single floor in a building. It defines the floorplan artwork, as well as all information relevant for localization (sections, walls, beacons, obstacles).

Section a section defines a region on the floor (bounded by a polygon) in which localization can be done.

Wall a wall defines a linear obstacle for movement inside a Section. Obstacle an obstacle defines a region in which no movement is allowed (e.g. a desk, closet, table, ...)

Beacon a yooBee beacon serves as an anchor point for localization. It may be a plain beacon or a basestation. The latter implements the handover between the yooBee BLE network and a TCP/IP network.

Mobile a mobile device that is localized, which can be either a tag or a phone/tablet (which uses the yooBee Mobile Phone SDK to relay positioning information back).

Protocol

Transport

The protocol uses HTTP, and uses JSON for data-exchange. For a cloud deployment the endpoints for the webservices point to Blooloc's servers, which are accessible only using secure HTTP:

https://api.blooloc.com/

For a local deployment, on a developer's system, the endpoint is available using HTTP over port 80 on the IP address assigned to the local system.

http://www.xxx.yyy.zzz/

The value of www.xxx.yyy.zzz depends on the local network configuration. In case of a Developer kit, if one accesses the system using the integrated access point, then the IP address is 10.0.0.1.

Versioning

The general request syntax is: `https://api.blooloc.com/version/path`

The *version* refers to an API version. This document corresponds to version *2.0*. When the *version* is omitted from the request, the version assumed is 0.1, for backward compatibility reasons.

Current versions available are:

Version	Date	Description
0.1	Nov. 2014	API pre-release
1.0	Nov. 2015	API release, **deprecated** since June 2017
2.0	May 2017	API v2 release

Within a single API version, fields may be added to results over time, but no existing fields or calls will ever be removed.

Authentication

Authentication is implemented using the OAuth 2 protocol (RFC 6749). The Positioning Server acts as a protected Resource and Authorization Server.

All interaction through Web Services require a valid OAuth 2 Bearer token. Each affiliation has its own token, which is available from within the web console of the user interface.

The OAuth2 authentication token using Bearer token works by including the token inside each request, in the HTTP Authorization header field.

For example (for a bearer token 'abcde-xyz-1234'):

```
GET /2.0/infrastructure/mobiles/list HTTP/1.1
Host: https://api.blooloc.com/
Authorization: Bearer abcde-xyz-1234
```

In the future, we anticipate a more fine-grained access control system. As an administrator for your affiliation, you will have the ability to manage a set of tokens and associate with each token more specific access control, narrowing down access to API features and positioning scope in terms of venues and mobiles.

GET requests: a Practical Example

The following request is often seen throughout the manual:

```
GET /url/request/path
Parameters:
- One: foo
- Two: bar
```

Multiple browser extensions are available to facilitate sending this request. Examples are ModHeader (Chrome extension), Postman (Chrome app), Modify-Header (Firefox Add-on) and REST Easy (Firefox Add-on). The url with parameters for this example will then be the following:

https://api.blooloc.com/url/request/path?One=foo&Two=bar

And the correct header containing the bearer token:

Header field name	Example value
Authorization	Bearer abcde-xyz-1234

A missing or incorrect bearer token in the header will result in a *401* error with the following JSON response:

```
{
  "error-code": 401,
  "error-message": "Incorrect auth bearer token"
}
```

Example response body:

```
[
  {
    "address": {
      "city": "Herent",
      "country": "Belgium",
      "name": "BlooLoc office",
```

```
      "postal-code": "3020",
      "state": "",
      "street": "Duigemhofstraat 101"
    },
    "id": 52
  },
  {
    "address": {
      "city": "Gotham",
      "country": "Tommorrowland",
      "name": "Smart Systems HQ",
      "postal-code": "JPG 1234",
      "state": "",
      "street": "Intelligent Street 007"
    },
    "id": 8
  },
  ...
]
```


PUT request: a Practical Example

PUT request, used to create or update objects, can be performed easily using e.g. the Postman or REST Easy plug-ins. In the body of the request, data is added in JSON. Online tools for JSON-validation can be useful to check for syntax errors before sending any request.

The server will handle your request and, if successful, will send a response back, often containing the id of the newly created object.

The header of a PUT request should, like the GET request, contain a Authorization token. The request body is a JSON object:

```
{
  "venue-id": 62,
  "name": "test venue",
  "elevation": 9001,
  "floorplan": {
    "floorplan-images": [],
    "beacons": [],
    "sections": [],
    "obstacles": [],
    "walls": [],
    "grids": [],
    "minGridSpacing": 0.5,
    "selectedGridAngle": 0,
    "selectedImageIndex": 0
  }
}
```

The response of this successful request is the floor id:

```
{
  "id": 353
}
```

On failure, the response contains an error code and message. For example, when trying to add a floor with the same name twice:

```
{
  "error-code" : 400,
```

```
  "error-message" : "Floor with name \"test venue\" already exists within the current venue"
}
```

Infrastructure API

This section describes the APIs that allow a user to retrieve information on venues to which he has access, and information on the roll out of yooBee hardware in those venues.

The infrastructure definition can also be modified using these APIs. This is mostly useful for the definition of floorplan information for venues, allowing this information to be automatically imported from a GIS platform.

Venues

There are two overlapping APIs to access venue definition data. The calls below describe a venue as a whole, while a set of APIs described further (see Floors) offer a floor-centric view and editing capabilities.

GET /2.0/infrastructure/venues/list Returns a Collection of VenueInfo objects (based on current access permissions) which is intended to build a list of venues available to the user.

GET /2.0/infrastructure/venues/show/:id Returns information of a venue identified by the id parameter. The call returns a Venue object with the following fields:

Parameters: none

Table 2.3: *VenueInfo* object

Field	Type	Description
id	Int64	ID (unique auto-generated number)
address	*Address*	The venue address
coordinates	*LatLng*	Optional GPS coordinates of the (origin of) the venue.

Table 2.4: *Venue* object

Field	Type	Description
id	Int64	ID (unique auto-generated number)
owner-id	Int64	Owner ID
address	*Address*	The venue address
global-transform	*Transformation*	An optional transformation matrix which converts the meter coordinate system into GPS coordinates (WGS84).

Field	Type	Description
coordinates	*LatLng*	GPS coordinates of the (origin of) the venue.
floors	Collection of *Floor*'s	A list of floors, ordered from bottom to top (on elevation).

Table 2.5: *Address* object

Field	Type	Description
name	String	Venue name (e.g. company)
street	String	Street name and number
postal-code	String	Postal code
city	String	City
state	String	State
country	String	Country

Table 2.6: *Floor* object

Field	Type	Description
id	Int64	ID (unique auto-generated number)
name	String	A floor name (e.g. "0" or "1.5"), which is unique for the venue
elevation	Float	Elevation of the floor relative to street level.
floorplan-images	Collection of *Image*	Floorplan (background) images.
beacons	Collection of *Beacon-Placement*	Beacons located on the floor.
sections	Collection of *Section*	Sections are areas of possible movement.
obstacles	Collection of *Obstacle*	Obstacles for movement.
walls	Collection of *Wall*	Walls (linear obstacles for movement).

Table 2.7: *Transformation* object

Field	Type	Description
matrix	Collection of Floats	6 floats representing a 2D transformation matrix: m11, m21, m22, dx and dy

Table 2.8: *LatLng* object

Field	Type	Description
latitude	Float	Latitude, in degrees
longitude	Float	Longitude, in degrees

Table 2.9: *Image* object

Field	Type	Description
name	String	An optional name for the image.
url	String	URL to the image (e.g. a PNG).
scale	Float	Scale for the image, in meters per pixel.
origin	*Coordinates*	Origin coordinates of the image in pixel.

Table 2.10: *Section* object

Field	Type	Description
id	Int64	ID (auto-generated number) unique within a floor.
name	String	An optional name.
description	String	An optional description.
linkout-id	String	An optional ID used to link to the customer data model.
type	String	A section type, which may be *none*, *room*, *hallway*, *elevator*, or *staircase*.
boundary	Collection of *Coordinates*	Vertices that describe a closed polygon.

Table 2.11: *Obstacle* object

Field	Type	Description
id	Int64	ID (auto-generated number) unique within a floor.
name	String	An optional name.
description	String	An optional description.
boundary	Collection of *Coordinates*	Vertices that describe a closed polygon.

Table 2.12: *Wall* object

Field	Type	Description
id	Int64	ID (auto-generated number) unique within a floor.
p1	*Coordinate*	First end-point
p2	*Coordinate*	Second end-point
width	Float	Wall thickness

Table 2.13: *Coordinate* object

Field	Type	Description
x	Float	X coordinate, in meters
y	Float	Y coordinate, in meters
z	Float	Z coordinate, in meters

Table 2.14: *Beacon-Placement* object

Field	Type	Description
deployment-id	Int64	Deployment ID (auto-generated number) unique within a floor.
position	*Coordinate*	Location on the floor, relative to the floor origin (the Z component reflects the beacon's height relative to the floor).
direction	*Coordinate*	Direction in which it is oriented (a normalized unity vector).
type	String	Beacon type ("beacon" or "basestation").

Example VenueInfo object:


```
{
  "address" : {
    "city" : "Herent",
    "country" : "Belgium",
    "name" : "BlooLoc office",
    "postal-code" : "3020",
    "state" : "",
    "street" : "Duigemhofstraat 101"
  },
  "floors" : [
    {
      "beacons" : [
        {
          "deployment-id" : 1,
          "direction" : {
            "x" : 0.999990640124791,
            "y" : 0,
            "z" : 0.004326622564088
          },
```

```
      "enabled" : false,
      "position" : {
        "x" : 0.220465677579009,
        "y" : 18.928289536021364,
        "z" : 2.3
      },
      "type" : "beacon"
    },
    ...
  ],
  "elevation" : 7.5,
  "floorplan-images" : [
    {
      "name" : "Blooloc Floorplan.jpg",
      "origin" : {
        "x" : 0,
        "y" : 0
      },
      "scale" : 0.006603683228917,
      "url" : "https://url/to/floorplan.jpg"
    }
  ],
  "id" : 80,
  "name" : "HQ 1nd floor",
  "obstacles" : [
    {
      "boundary" : [
        {
          "x" : 3.5,
          "y" : 8.75
        },
        {
          "x" : 6.75,
          "y" : 8.75
        },
        {
          "x" : 6.75,
          "y" : 7.5
        },
        {
          "x" : 3.5,
          "y" : 7.5
        }
      ],
      "description" : "",
      "id" : 1,
      "name" : ""
    },
    ...
  ],
  "sections" : [
    {
      "boundary" : [
        {
          "x" : 2.5,
          "y" : 4.25
        },
        {
          "x" : 6.5,
          "y" : 4.25
        },
        {
          "x" : 6.5,
          "y" : 0.5
        },
        {
          "x" : 2.5,
          "y" : 0.5
```

```
          }
        ],
        "description" : "Innovation Meeting room",
        "id" : 1,
        "linkout-id" : "",
        "name" : "Innovation",
        "type" : "room"
      },
      ...
    ],
    "walls" : [
      {
        "id" : 1,
        "p1" : {
          "x" : 9.25,
          "y" : 15.5
        },
        "p2" : {
          "x" : 9.25,
          "y" : 19.25
        },
        "width" : 0.2
      },
      ...
    ]
  }
 ],
 "id" : 52,
 "owner-id" : 29
}
```


GET /2.0/infrastructure/venues/search Searches venues based on criteria. The only criterion currently considered is a beacon hardware ID (MAC or iBeacon address).

This returns a collection of Venue objects.

Parameters:

beacon-id required A comma separated list of beacon hardware IDs (MAC or iBeacon address) or the parameter can be listed multiple time within the request for multiple values.

Floors

These APIs provide information overlapping with the APIs that provide venue information, however they also allow floor definitions to be added or updated.

GET /2.0/infrastructure/floors/list Returns a Collection of Floor objects associated with your affiliation.

Parameters: none

GET /2.0/infrastructure/floors/show/:id Returns information of a floor identified by the (database) id parameter. The call returns a Floor object.

Parameters: none

PUT /2.0/infrastructure/floors/new adds a floor

This adds a new floor. The body must contain a *Floor-update* object. The 'id' field of *Beacon-Placement*, *Section*, *Obstacle* and *Wall* objects is ignored as these values are provided by the server.

Parameters: none

Table 2.15: *Floor-update* object

Field	Type	Description
venue-id	Int64	References the venue to which the floor will be added
name	String	A floor name, which is unique within the venue
elevation	Float	Elevation of the floor relative to street level (in meter)
floorplan	*Floorplan*	The floorplan information

The floorplan background image, as specified as an Image object, contains an URL that specifies the image contents within a data URL.

Table 2.16: *Floorplan* object

Field	Type	Description
floorplan-images	Collection of *Image*	Floorplan (background) images
beacons	Collection of *Beacon-Placement*	Beacons located on this floor
sections	Collection of *Section*	Sections are areas of possible movement
obstacles	Collection of *Obstacle*	Obstacles for movement
walls	Collection of *Wall*	Walls (linear obstacles for movement)
grids	Collection of *Grid*	Grids
minGridSpacing	Float	Minimum grid spacing
selectedGridAngle	Float	The selected rotation angle
selectedImageIndex	Int64	Index of the selected floorplan Image

Table 2.17: *Grid* object

Field	Type	Description
angle	Float	The rotation angle

Example PUT request for `/2.0/infrastructure/floors/new`:

```
{
  "venue-id": 62,
  "name": "test venue",
  "elevation": 9001,
```

```
  "floorplan": {
    "floorplan-images": [],
    "beacons": [],
    "sections": [],
    "obstacles": [],
    "walls": [],
    "grids": [],
    "minGridSpacing": 0.5,
    "selectedGridAngle": 0,
    "selectedImageIndex": 0
  }
}
```

The response of this successful request is the floor id:

```
{
  "id": 353
}
```

POST /2.0/infrastructure/floors/:id updates the information of a floor

This updates an existing floor, identified using its database ID. The body contains a *Floor-update* object. The 'id' field of *Beacon-Placement*, *Section*, *Obstacle* and *Wall* objects is ignored as this value is provided by the server.

Sending an empty collection of *Beacon-Placement*, *Image*, *Section*, *Obstacle* or *Wall* objects will delete these objects from the floor. Omitting these objects (or providing Null values for them) in the *Floor-update* object will on the other hand retain the old objects. This can be used to update only specific aspects of a floor.

Parameters:: none

Beacons

A separate set of APIs can be used to query information about beacon devices, and register new devices (although beacons will also automatically be registered if they are being detected by any of your basestations).

GET /2.0/infrastructure/beacons/list Returns a Collection of Beacon devices associated with your affiliation.

Parameters: none

Table 2.18: *Beacon* object

Field	Type	Description
id	Int64	ID (unique auto-generated number) that uniquely identifies the beacon device.
venue-id	Int64	For a deployed beacon, references the venue in which it is deployed.
floor-id	Int64	For a deployed beacon, references the floor in which it is deployed.

Field	Type	Description
deployment-id	Int64	For a deployed beacon, matches a *Beacon-Placement* object on the floor.
description	String	An optional description for the beacon.
enabled	boolean	Indicates whether the beacon is in use.
last-seen	Timestamp	Time when last seen.
type	String	Beacon type (*beacon* or *basestation*).
mac	String	BLE MAC address.
uuid	String	iBeacon address (UUID major minor).
firmware	String	Current firmware version.
basestation-id	String	For a normal beacon, the ID of the basestation through which it last communicated.
affiliation-id	String	For a normal beacon, the ID of the affiliation.

GET /2.0/infrastructure/beacons/show/:id Returns information of a beacon identified by the (database) id parameter. The call returns a *Beacon* object.

Example BeaconInfo object:


```
[
  {
    "affiliation-id": 4,
    "basestation-id": 1234,
    "deployment-id": 20,
    "description": "AB-001",
    "enabled": true,
    "firmware": "66864fe_57f30cb",
    "floor-id": 666,
    "id": 2107,
    "last-seen": "2016-10-14T11:01:24",
    "mac": "ab:12:cd:34:1c:ee",
    "type": "beacon",
    "uuid": "3f4fadf9-abcd-1234-a12b-beba57938ad8 ee1c 84e5",
    "venue-id": 133
  },
  ...
]
```


GET /2.0/infrastructure/beacons/search Searches beacons according to a number of search criteria. The result is a list of Beacon objects.

Parameters:

mac optional Search on MAC address

PUT /2.0/infrastructure/beacons/new adds a beacon

This adds a new beacon. The body must contain a *Beacon* object. The 'id', 'last-seen' and 'UUID' fields are ignored as these values are provided by the server.

Parameters: none

PUT /2.0/infrastructure/beacons/:id updates information of a beacon

This updates an existing beacon, identified using its database ID. The body contains a

Beacon object. The 'id', 'last-seen' and 'UUID' fields are ignored as these values are provided by the server.

Mobiles

Likewise, a set of APIs can be used to query information about mobile devices (both tags and smartphones/tablets), and register new devices (although smartphones and tags will also automatically be registered on first use).

Information on mobile devices that are registered with an affiliation may be requested and updated using a set of APIs. In this context, mobile devices are both phones/tablets or yooBee tags.

GET /2.0/infrastructure/mobiles/list Returns a Collection of *Mobiles*.

Parameters: none

Table 2.19: *Mobile* object

Field	Type	Description
id	Int64	ID (unique auto-generated number).
assignee-name	String	An optional name for the assignee of the mobile.
type	String	A mobile type which may be one of tag, or smartphone.
device-id	String	A device ID.
description	String	Description of mobile.
last-seen	Timestamp	Time when last seen.
linkout-id	String	An ID used to link to the customer data model.
movement-type	String	Either "walking" or "riding" depending on the mobile movement type

For a yooBee tag device, the device-id is the BLE MAC address of the device. For a phone or tablet, the device-id is the ID with which the phone identifies itself to the system.

Example Mobile object (tag):


```
[
  {
    "assignee-name": "Lise",
    "description": "Lise Tag",
    "device-id": "cb:fb:85:1f:2a:d9",
    "id": 915,
    "last-seen": "2016-10-10T11:28:49",
    "linkout-id": "",
    "movement-type": "walking",
    "type": "tag",
  },
  ...
]
```

GET /2.0/infrastructure/mobiles/show/:id Returns information of a mobile identified by the (database) id parameter. The call returns a *Mobile* object.

GET /2.0/infrastructure/mobiles/search Searches mobiles according to a number of search criteria. The result is a list of *Mobile* objects.

Parameters:

type optional Restricts the type (tag or phone)

device-id optional A comma delimited list of device IDs or the parameter can be listed multiple times within the request for multiple values.

description optional A comma delimited list of descriptions or the parameter can be listed multiple times within the request for multiple values.

linkout-id optional A comma delimited list of linkout IDs or the parameter can be listed multiple time within the request for multiple values.

PUT /2.0/infrastructure/mobiles/new adds a mobile

This adds a new mobile. The body contains a *Mobile* object, whose data is used to populate the database. The 'last-seen' field is ignored since it's automatically computed by the system.

PUT /2.0/infrastructure/mobiles/:id updates information for a mobile

This updates an existing mobile, identified using its database ID. The body contains a Mobile object, whose data is used to populate the database. The 'last-seen' field is ignored since it's automatically computed by the system.

Positioning API

This is a set of APIs to retrieve positioning results. Positioning results may be queried in terms of coordinates ('locations'). A first set of APIs allows one to retrieve historical positioning results, while a second set of APIs allows one to stream the latest positioning updates (for real-time processing).

Since API v2.0, the venue-id parameter became mandatory when streaming or searching retrieving for positioning data. Data for associated venues (which share the license) are streamed together with their primary venue.

Querying

GET /2.0/positioning/locations/search Searches historical positioning results according to a number of search criteria. The result is a list of *Location* objects.

The search can be narrowed down in terms of mobiles, but also in terms of the location in which to search.

The query needs a *venue-id* and a time window to be specified.

This API call should not be used to poll positioning data using time=last as this generates a high load on the server. Instead, use the streaming APIs which are more efficient and guarantee to provide all data in real-time.

Parameters:

venue-id required A venue id. Exactly one venue-id is required.
time required A single unix timestamp format or UTC format e.g. *2015-01-30T10:22:33*, a time range (*from,to*) or *last*
mobile-id optional A comma-separated list of mobiles database ids or the parameter can be listed multiple times within the request for multiple values.
mobile-device-id optional This parameter is used as an alternative for **mobile-id** and should not be used at the same time. The parameter has to be listed multiple times within the request for multiple values e.g. *&mobile-device-id=aa:bb:cc:dd:ee:ff&mobile-device-id=MyPhone* and represents the mobiles device id.
quality optional quality of the position (0 realtime, 1 delayed)
floor-id optional A comma-separated list of floors database ids (for one venue being searched) or the parameter can be listed multiple times within the request for multiple values.
(position) optional A comma-separated list of positions, each in the format "(x, y)" (for one floor being searched) or the parameter can be listed multiple times within the request for multiple values.
(search-radius) optional The search radius around the given positions
coord-system optional Reference coordinate system for the position: *localmeters* (default) or *wgs84*.

Example Location object:

```
GET parameters:
- time = 2016-11-17T10:20:20
- venue-id = 142
```


```
[
  {
    "floor-id": 271,
    "heading": -1.6414184877112,
    "mobile-id": 3033,
    "position": {
      "X": -4.97888785965827,
      "Y": 9.37499583348057
    },
    "position-accuracy": {
      "X": 1.24521109867192,
      "Y": 0.364031962562621
    },
    "quality": 1,
```

```
    "state": "active",
    "time": 1479378020,
    "venue-id": 142,
    "walking": 0.003516977141003
  },
  ...
]
```

Table 2.20: *Location* object

Field	Type	Description
mobile-id	Int64	mobile logical ID
time	Timestamp	Unix timestamp for this location
quality	Int64	quality of the position (0 realtime, 1 delayed)
venue-id	Int64	venue ID
floor-id	Int64	floor ID
position	*Coordinate* or *LatLng*	Position, type depends on *coord-system* request parameter. (Not present in case of an error in positioning.)
position-accuracy	*Coordinate*	Position standard deviation
heading	Float	Heading (clockwise), in radians, (0 is x-axis). For WGS84, heading corresponds to the geographic direction (0 is north). (Not present in case of an error in positioning.)
walking	Float	Probability that the mobile is walking/moving
state	String	A state, which may be *exited* (the mobile exited the building or was switched off), *active* (location is known and the tag has recently moved), or *inactive* (location is known but the tag hasn't recently moved).
error	String	A string containing an info message in case of errors in positioning (e.g. requesting lat/long positioning while GPS is not setup properly).

Streaming

Streaming APIs provide an efficient way to connect to a stream to receive real-time updates on positioning[1].

The streaming API accepts the same parameters as the counter-part querying API, with the exception of *time* which is not available for a streaming API.

Note: If you're testing the API with a mock-up client, then please check first if streaming is supported by this client. Streaming does not appear to work in the Postman extension, but a browser should work properly.

[1] as in https://dev.twitter.com/docs/streaming-apis/processing

GET /2.0/positioning/locations/stream stream location updates for one or more mobiles

The API has an additional parameter 'delimited=' which controls the delimitation between different results. The following values are supported:

(default) by default, JSON packets are delimited by "\r\n\r\n"

delimited=length each JSON packet is preceded by a string representation of a base-10 integer indicating the length of the message in bytes

delimited=line each JSON packet is one line, delimited by "\r\n".

Default example stream:


```
{
    "floor-id" : 215,
    "heading" : 2.895736847368416,
    "mobile-id" : 207,
    "position" : {
        "X" : 23.155663367194958,
        "Y" : -5.355774436569296e00
    },
    "position-accuracy" : {
        "X" : 0.093786244987011,
        "Y" : 0.084021596633497
    },
    "quality" : 0,
    "state" : "active",
    "time" : 1473831290,
    "venue-id" : 119,
    "walking" : 2.333721519591079e-24
}

{
    "floor-id" : 215,
    "heading" : 2.820441590690582,
    "mobile-id" : 207,
    "position" : {
        "X" : 23.139877526273732,
        "Y" : -5.360675981131615e00
    },
    "position-accuracy" : {
        "X" : 0.064509831334131,
        "Y" : 0.096354875867342
    },
    "quality" : 1,
    "state" : "active",
    "time" : 1473831276,
    "venue-id" : 119,
    "walking" : 5.965229201196621e-22
}
...
```


Length delimited example stream:


```
333{
    "floor-id" : 215,
    "heading" : -1.257522156642102e00,
    "mobile-id" : 207,
    "position" : {
        "X" : 13.663158031073532,
        "Y" : -9.682261828849418e00
    },
    "position-accuracy" : {
        "X" : 0.259889803456418,
```

```
        "Y" : 0.335506807469299
    },
    "quality" : 0,
    "state" : "inactive",
    "time" : 1473830494,
    "venue-id" : 119,
    "walking" : 0
}329{
    "floor-id" : 215,
    "heading" : 0.802653942378847,
    "mobile-id" : 207,
    "position" : {
        "X" : 13.644832386426328,
        "Y" : -9.641539733367081e00
    },
    "position-accuracy" : {
        "X" : 0.205291093272407,
        "Y" : 0.247274897433464
    },
    "quality" : 1,
    "state" : "inactive",
    "time" : 1473830479,
    "venue-id" : 119,
    "walking" : 0
}...
```


Line delimited example stream:


```
{ "floor-id" : 215, "heading" : 3.121171277157191, "mobile-id" : 201, "position" : { "X" : ..., "Y" : ... }, ...}
{ "floor-id" : 215, "heading" : 3.109372168719986, "mobile-id" : 201, "position" : { "X" : ..., "Y" : ... }, ...}
{ "floor-id" : 215, "heading" : 3.132451883268586, "mobile-id" : 201, "position" : { "X" : ..., "Y" : ... }, ...}
{ "floor-id" : 215, "heading" : 3.134618120475023, "mobile-id" : 201, "position" : { "X" : ..., "Y" : ... }, ...}
...
```


Event API

This is a set of APIs to retrieve information about the occurrence of certain events. A first set of APIs allows one to retrieve historical events, while a second set of APIs allows one to obtain a live stream of events.

Since API v2.0, exactly one venue-id is required when searching or streaming event data.

Additional Streaming parameters

The streaming API has an additional parameter 'delimited=' which controls the delimitation between different results, see Streaming.

Additionally, a `keep-alive=x` parameter is available which will tell the server to send a keep-alive event every x seconds. This heartbeat can be useful for streams where the connection might close due to inactivity

Field	Type	Description

Table 2.21: *Keep-alive EventInfo* object

Field	Type	Description
event-type	String	For a keep-alive event: “keep-alive”.

Button Press

Querying

GET /2.0/events/button-press/search Searches button press events according to a number of search criteria. The result is a list of EventInfo objects.

The search can be narrowed down in terms of mobiles and the query needs a time window to be specified.

Parameters:

venue-id required A venue id. Only one venue-id is allowed.
time required A single unix timestamp format or UTC format e.g. *2015-01-30T10:22:33*, a time range (*from*,*to*) or *last*
mobile-id optional A comma-separated list of mobiles database ids or the parameter can be listed multiple times within the request for multiple values.

Table 2.22: *Button-press EventInfo* object

Field	Type	Description
event-type	String	For button press events, this corresponds to “button-press”
id	Int64	Database ID of the mobile
time	Timestamp	Unix timestamp for this event
venue-id	Int64	Optional. The venue id of the mobile.

Streaming

The streaming API provides an efficient way to connect to a stream to receive live updates on button presses.

GET /2.0/events/button-press/stream Stream button press updates for one or more mobiles

The streaming API accepts the same parameters as the querying API with the exception of time which is not available for a streaming API.

Status API

This section describes the APIs that allow a user to retrieve information on the current status of the various devices (mobiles, beacons, basestations). These APIs are useful to monitor the correct operation of the system, and to raise events related to maintenance.

Note: For a demo-kit, the status APIs are not hosted on port 80 but rather on port 8083.

Beacons

GET /2.0/status/devices/beacons Returns a Collection of BeaconStatus objects (based on current access permissions).

Parameters:

venue-id required A comma separated list of venue IDs.
id optional A comma separated list of beacon IDs.
status optional *connected* or *disconnected*

Table 2.23: *BeaconStatus* object

Field	Type	Description
id	Int64	ID (unique auto-generated number) that uniquely identifies the beacon device
hardware-id	String	BLE MAC address
battery	Float	Charge of the battery (V)
last-battery	Timestamp	Time at which the last battery measurement was obtained.
charge-state	Float	Charge-state of the battery (mA); this does not necessarily reflect what the solar panel harvests, but what the battery absorbs: it may be clipped if the battery is full, for example
comm-quality	*TimeStats*	A structure that contains a measure for communication quality (a number between 0 and 1 where 0 means no communication, and 1 means perfect communication) averaged over different time scales
last-comm	Timestamp	Time at which the beacon last communicated through a basestation
last-seen	Timestamp	Time when the beacon was last seen by a mobile (could be a phone/tablet too)

Field	Type	Description
basestation-hardware-id	String	BLE MAC address of the basestation through which this beacon has last communicated
status	String	Connection state (*connected* or *disconnected*)
description	String	Description of the Beacon.

Table 2.24: *TimeStats* object

Field	Type	Description
d0 0:01	Float	Value averaged over the last minute.
d0 0:05	Float	Value averaged over the last five minutes.
d0 1	Float	Value averaged over the last hour.
d1	Float	Value averaged over the last day.
d7	Float	Value averaged over the last week.
t	Int64	UNIX timestamp of last update.

Basestations

GET /2.0/status/devices/base-stations Returns a Collection of BasestationStatus objects (based on current access permissions).

Parameters:

venue-id required A comma separated list of venue IDs.
id optional A comma separated list of basestation IDs.
status optional *connected* or *disconnected*

Table 2.25: *BasestationStatus* object

Field	Type	Description
id	Int64	ID (unique auto-generated number) that uniquely identifies the basestation device
hardware-id	String	BLE MAC address
comm-quality	*TimeStats*	A structure that contains a measure for communication quality (a number between 0 and 1 where 0 means no communication, and 1 means perfect communication) of the basestation with beacons, averaged over different time scales
last-comm	Timestamp	Time at which the basestation last communicated
last-connect	Timestamp	Time when the basestations last connected
status	String	Connection state (*connected* or *disconnected*)
beacon-count	Short	Number of beacons connected to the basestation

Field	Type	Description
tag-count	Short	Number of tags connected to the basestation
eth-IP-address	String	IP address assigned to the Ethernet port
eth-MAC-address	String	MACC address of the Ethernet port
wifi-IP-address	String	IP address assigned to the WIFI access point or client
wifi-SSID	String	SSID of the WIFI access point
description	String	Description of the Basestation.

Tags

GET /2.0/status/devices/tags Returns a Collection of TagStatus objects (based on current access permissions).

Parameters:

venue-id required A comma separated list of venue IDs
id optional A comma separated list of beacon IDs

Table 2.26: *TagStatus* object

Field	Type	Description
id	Int64	ID (unique auto-generated number) that uniquely identifies the tag
hardware-id	String	BLE MAC address
comm-quality	*TimeStats*	A structure that indicates the average number of beacons communicating with the tag, averaged over different time scales
last-comm	Timestamp	Time at which the tag last communicated
compass-status	*TimeStats*	A structure that indicates the stability of the compass as an entropy measure (a number between 0 and 1, lower is better), averaged over different time scales
estimated-P0	*TimeStats*	A structure that indicates the estimated P0 of the tag (dBm), averaged over different time scales
heading-status	*TimeStats*	A structure that indicates the entropy of raw tag heading measurements (a number between 0 and 1, higher is better), averaged over different time scales
localization-health	*TimeStats*	A structure that indicates the length in moving time (s) during which the tag was continuously being tracked (higher is better), averaged over different time scales

Field	Type	Description
battery	Float	Charge of the battery (V)
last-battery	*TimeStats*	Time at which the last battery measurement was obtained.
venue-id	Int64	(Optional) ID of the venue the tag is currently seen in.