LBD Backend Documentation

Release Dev

LBD Development

January 14, 2015

CONTENTS

Table of Contents

CHAPTER

ONE

REQUIREMENTS

The back-end is written using Python 2.7. While it is considered to be legacy and (if we are mean) old, it is quite widely spread and common version of Python. The good thing is that it is stable and does not change anymore, while Python 3 is still in active development and has not yet seen the final form. Maybe at some point, if this software is developed further, we will move from 2.7 to 3.x :) .

As database we decided to try out MongoDB, a NoSQL database that stores information as JSON documents. The reason for this was to try out something other than SQL and because MongoDB has support for geospatial data builtin, which made the use of GeoJSON easier. In addition we could also form geospatial indexes and searches without any additional plugins. To use the MongoDB database from Python code, mongoengine and pymongo libraries are used. These can be installed with Pip.

Software requirements:

- **MongoDB 2.6.6** or greater (designed with 2.6.6)
- **Python 2.7.x** (designed with 2.7.8)
  - **pymongo 2.7.2** or greater (designed with 2.7.2) (*Python library*)
  - **mongoengine 0.8.7** or greater (designed with 0.8.7) (*Python library*)
  - **httplib2 0.9** or greater (designed with 0.9) (*Python library*)

CHAPTER

TWO

INSTALLATION

This chapter describes where one can download the software and libraries required by the back-end and how they can be installed.

2.1 MongoDB

MongoDB can be downloaded from http://www.mongodb.com/downloads for different operating systems. For Linux distributions it might be available from package management system (e.g. Portage on Gentoo) depending on the architecture running the operating system.

MongoDB installation guides for different operating systems can be found from http://docs.mongodb.org/manual/installation/ .

2.2 Python

Python can be downloaded from https://www.python.org/downloads/ for multiple operating systems. As with MongoDB, on Linux it can also be installed with package manager.

2.2.1 Pip

PIP is recommended for easy installation of Python libraries. Now for those using the latest versions of Python, there are some good news, as PIP comes with the installation and on Linux of course it might be installed by the package manager. However, if you are not lucky enough to have **Python 2.7.9** or running Linux system, you need to do this yourself.

So... if you have Python <= 2.7.8 or otherwise do not have PIP yet, follow the instructions at https://pip.pypa.io/en/latest/installing.html .

After the installation, the latest versions of libraries can be installed by running:

```
pip install <package name>
```

or specific version:

```
pip install <package name>==x.y.z
```

2.3 Django

The recommended way to install django in all operating systems is through PIP:

```
pip install django
```

Of course nothing prohibits one to install it with Linux package manager or otherwise. More on django installation at https://docs.djangoproject.com/en/1.7/topics/install/ .

CHAPTER

THREE

HEADERS AND CORS

3.1 Headers

The back-end uses two un-standard headers for user authentication and authorization. Both of these are required for most functionalities.

Header name	Explanation
LBD_LOGIN_HEADER	Google OAuth authentication token.
LBD_OAUTH_ID	Google id.

3.2 CORS

CORS or Cross-origin resource sharing mechanism allows resources to be requested from other domains. In order to be able to use the back-end for example with AngularJS from another domain, some headers needed to be added. This is done in cors-middleware.

Cors middleware adds support for HTTP OPTIONS method and adds *Access-Control-Allow-Origin*, *Access-Control-Allow-Credentials* and *Access-Control-Allow-Headers* to the request.

CHAPTER

FOUR

JSON FORMATS

4.1 Location data

Location data uses GeoJSON (http://geojson.org/) specification, so all objects retrieved from the back-end are compatible with GeoJSON readers. Thanks to GeoJSON's flexibility, new elements can be added to the objects. Back-end utilizes this by adding "metadata" element inside "properties" element. This new field is completely optional and may contain information such as status of the location data object, who has modified the metadata and so on. In this chapter this new element is described in detail.

4.1.1 JSON Format

Like stated above, the JSON used by the software follows the GeoJSON specification.

An example from http://gejson.org/geojson-spec.html (referenced 11.1.2015):

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [102.0, 0.5]
            },
            "properties": {
                "prop0": "value0"
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "LineString",
                "coordinates": [
                    [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
                ]
            },
            "properties": {
                "prop0": "value0",
                "prop1": 0.0
            }
        },
        {
            "type": "Feature",
```

```
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                     [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
                       [100.0, 1.0], [100.0, 0.0] ]
                 ]
            },
            "properties": {
                "prop0": "value0",
                "prop1": {
                    "this": "that"
                }
            }
        }
    ]
}
```

Note: It should be noted that the current system does not support other feature types than "Point".

Back-end supports GeoJSON Feature when relaying information on single object and FeatureCollection when sending multiple objects.

The format for the additional information is:

Field's name	Required	Value type	Notes
status	True	String	
modified	True	String	Unix timestamp (in seconds). Tells when the metadata was modified
modifier	True	String	Tells who has modified the metadata
info	True	String	

Which translates to:

```
"metadata": {
    "status": **String**,
    "modified": **Integer**,
    "modifier": **String**,
    "info": **String**
}
```

And a "real" example using open data service of Tampere (http://www.tampere.fi/tampereinfo/avoindata.html) to provide the Streetlight information:

```
{
    "geometry": {
        "type": "Point",
        "coordinates": [23.643239226767022, 61.519112683582854]
    },
    "id": "WFS_KATUVALO.405172",
    "type": "Feature",
    "properties": {
        "NIMI": "XPWR_6769212",
        "LAMPPU_TYYPPI_KOODI": "100340",
        "TYYPPI_KOODI": "105007",
        "KATUVALO_ID": 405172,
        "LAMPPU_TYYPPI": "ST 100 (SIEMENS)",
        "metadata": {
            "status": "foobar",
            "note": "FOOBAR",
```

```
            "modifier": "tiina@teekkari.fi",
            "modified": 1420741774
        }
    },
    "geometry_name": "GEOLOC"
}
```

4.2 Message

The JSON for messages has been influenced by GeoJSON (a little bit). This is visible when comparing how single and multiple messages are relayed. A single message has type "Message" while multiple messages are sent in a JSON "envelope" with "MessageCollection" as type.

An example containing both single message and a collection of messages:

```
{
    "messages": [
        {
            "category": "Streetlights",
            "attachments": [
                {
                    "category": "Streetlights",
                    "id": "WFS_KATUVALO.405172"
                }
            ],
            "type": "Message",
            "topic": "Testi",
            "messageread": true,
            "message": "Tämä on testiviesti",
            "recipient": "tiina@teekkari.com",
            "id": 10,
            "sender": "tiina@teekkari.com"
        },
        {
            "category": "Streetlights",
            "attachments": [
                {
                    "category": "Streetlights",
                    "id": "WFS_KATUVALO.405172"
                }
            ],
            "timestamp": 1420786021,
            "topic": "Testi",
            "messageread": true,
            "type": "Message",
            "message": "Tämä on testiviesti",
            "recipient": "tiina@teekkari.com",
            "id": 11,
            "sender": "tiina@teekkari.com"
        }
    ],
    "type": "MessageCollection",
    "totalMessages": 2
}
```

Message collection has three elements:

Field's name	Required	Value type	Notes
type	True	String	Always "MessageCollection".
messages	True	List	List of messages. Format below.
totalMessages	True	Integer	Amount of messages in the collection.

Format for a single message is following:

Field's name	Required	Value type	Notes
type	True	String	Always "Message"
topic	True	String	Topic of the message
category	False	String	Category of the message. Name of a location data collection.
id	True	Integer	Message id.
sender	True	String	Tells who sent the message.
recipient	True	String	Tells who is the recipient.
message	True	String	Content of the message.
timestamp	True	Integer	Unix timestamp (in seconds). Tells when the message was sent.
attachments	False	List	List of attachments. Format described below.

Note: If the a category is specified for a message, there must exist a location data collection with that name.

If attachment element is added to a message, message category becomes a required field. For flexibility, the message category and the attachment category can be different from each other.

Attachment elements are JSON documents with two fields:

Fieldname	Required	Value type	Notes
category	True	String	Category of the attachment. Name of a location data collection.
id	True	String	Id of the attached object. Must exist in the specified location data collection.

4.3 Search

Search from the back-end is done by posting a JSON document to the back-end (see: *REST documentation*).

Field's name	Required	Value type	Notes
from	True	String	Only allowed value currently is "ALL"
search	True	String	Search phrase
limit	True	Integer	Maximum number of results returned.

Result format:

Field's name	Required	Value type	Notes
totalresults	True	Integer	The total amount of results.
limit	True	Integer	The defined limit of the results.
results	True	JSON	GeoJSON FeatureCollection containing the results.

Example:

```
SEARCH:
{
    "from": "ALL",
    "search": "42",
    "limit": 21
}

RESULT:
{
```

```
    "totalresults": 1138,
    "limit": 42,
    "results": { ...GeoJSON FeatureCollection... }
}
```

CHAPTER

FIVE

REST DOCUMENTATION

This chapter describes the REST APIs.

List of status codes used by the REST:

Status	Meaning	Notes
200	OK	Request successful
401	Unauthorized	"Login" failed.
403	Forbidden	You shall not pass!
404	Not Found	Resource not found.
405	Method not allowed	HTTP method not allowed.
418	I'm a teapot	Short and stout!
500	Internal Server Error	Server snafu'd

5.1 Location data REST

Location data can be accessed from **/locationdata/api/** (e.g. *www.example.com/locationdata/api/*). This URL does not yet require authentication and return the installed location data services and what data (element names) they contain.

Note: When creating or updating resources, only metadata is updated or created currently. It is not possible to create actual location data objects... yet.

In urls **<collection name>** and **<resource>** are to be replaced with appropriate values. Both are strings.

5.1.1 /locationdata/api/

Returns the installed location data services that can be accessed by appending the name of the service to the base url of the location data api.

Allowed methods:

- GET

URL parameters

- None

Example result:

```
[
    {
        "fields": [
            "geometry_name",
```

```
            "geometry.type",
            "geometry.coordinates",
            "id",
            "type",
            "properties.URAKKA_ALUE",
            "properties.OSA_ALUE_NIMI",
            "properties.PINTA_ALA",
            "properties.KAYTTOLK",
            "properties.ALUE_NIMI",
            "properties.TILAAJA",
            "properties.VIHERALUEEN_OSAN_ID",
            "properties.KAUPUNGINOSA",
            "properties.TOIMLK",
            "properties.ALUE_SIJ",
            "properties.HOITOLK"
        ],
        "name": "Playgrounds",
        "description": "Ring around the rosie"
    },
    {
        "fields": [
            "geometry_name",
            "geometry.type",
            "geometry.coordinates",
            "id",
            "type",
            "properties.NIMI",
            "properties.LAMPPU_TYYPPI_KOODI",
            "properties.TYYPPI_KOODI",
            "properties.KATUVALO_ID",
            "properties.LAMPPU_TYYPPI",
            "properties.TYYPPI"
        ],
        "name": "Streetlights",
        "description": "Tampere Streetlights"
    }
]
```

Note: The name element is the one to be added to the url.

5.1.2 /locationdata/api/<collection name>/

Allowed methods:

- GET
  - Returns the whole collection.
- DELETE
  - Deletes the whole collection.
- PUT
  - Replaces the collection.
- POST
  - Adds a new element to the collection.

URL parameters

- mini (*Optional*)
  - **Boolean** Returns minimum amount of data. Valid values: true or false

5.1.3 /locationdata/api/<collection name>/<resource>

Allowed methods:

- GET
  - Returns the resource.
- DELETE
  - Deletes the resource.
- PUT
  - Update or create a resource.

5.1.4 /locationdata/api/<collection name>/near/

Searches objects from circular area.

Allowed methods:

- GET
  - Returns the resources near the location.
- DELETE
  - Deletes the resources near the location.

URL parameters

- mini (*Optional*)
  - **Boolean** Returns minimum amount of data. Valid values: true or false
- latitude (*Required*)
  - **Float** The latitude of the circle's center
- longitude (*Required*)
  - **Float** The longitude of the circle's center
- range (*Optional*)
  - **Float** The radius of the circle

5.1.5 /locationdata/api/<collection name>/inarea/

Searches objects inside a rectangular area.

Allowed methods:

- GET
  - Returns the resources inside the area.

- DELETE
  - Deletes the resource inside the area.

URL parameters

- mini (*Optional*)
  - **Boolean** Returns minimum amount of data. Valid values: true or false
- xbottomleft (*Required*)
  - **Float** The longitude of the bottom left corner of the area.
- ybottomleft (*Required*)
  - **Float** The latitude of the bottom left corner of the area.
- xtopright (*Required*)
  - **Float** The longitude of the top right corner of the area.
- ytopright (*Required*)
  - **Float** The latitude of the top right corner of the area.

5.1.6 /locationdata/api/<collection name>/search/

Searches from the location data REST. Search is currently limited to the id.

Allowed methods:

- POST
  - Send the search JSON.

URL parameters

- None

5.2 Message data REST

The REST for sending messages in the system. For JSON formats, see *Message formats*

In URLs **<message id>** and **<category>** are to be replaced with appropriate values. Message id is an integer and category is a string.

5.2.1 /messagedata/api/send/

Allowed methods:

- POST
  - Send a message.

URL parameters

- None

5.2.2 /messagedata/api/users/list/

Lists all users.

Allowed methods:

- GET
  - Returns name and email of users.

URL parameters

- None

5.2.3 /messagedata/api/markasread/<message id>

Allowed methods:

- GET
  - Mark message read.

URL parameters

- None

5.2.4 /messagedata/api/messages/

Allowed methods:

- GET
  - Get user's all messages.
- DELETE
  - Delete user's all messages

URL parameters

- None

5.2.5 /messagedata/api/messages/<message id>

Allowed methods:

- GET
  - Get a single message.
- DELETE
  - Delete the message.

URL parameters

- None

5.2.6 /messagedata/api/messages/<category>/

Allowed methods:

- GET
  - Get user's all messages in certain category.
- DELETE
  - Delete user's all messages in certain category.

URL parameters

- None

5.2.7 /messagedata/api/messages/<category>/<message id>

Allowed methods:

- GET
  - Get a single message in a certain category.
- DELETE
  - Delete a single message in a certain category.

URL parameters

- None

CHAPTER

SIX

CODE DOCUMENTATION: REST

6.1 lbd_backend.LBD_REST_locationdata.decorators Location data decorators

6.1.1 Decorators for location data REST

This module contains the decorators for the REST handling the location data

lbd_backend.LBD_REST_locationdata.decorators.**lbd_require_login**(*func*)
Wrapper

This wrapper is used for authenticating the user with Google OAuth2.

Key "lbduser" is added to kwargs with User object as value.

lbd_backend.LBD_REST_locationdata.decorators.**location_collection**(*func*)
Wrapper

Checks if the collection in the URL exists and the handler for it is installed.

Key "handlerinterface" is added to kwargs with a handler object as the value.

6.2 lbd_backend.LBD_REST_locationdata Location data REST

6.2.1 View for handling the backend REST locationdata requests

This module handles http requests related to location data.

For all possible HTTP statuses, see *REST documentation*.

Status 200 is returned when request is valid and handled successfully while 400 is returned when the request conten is malformed or there is some other issues with the request..

Note: In case of PUT and POST, status 200 does not guarantee that any data has changed in database.

Status 400 is returned when request body does not match the defined format or there is some other inconsistency in the request.

Status 500 means that something went wrong when handling the request.

Client should be able to handle these responses and should not crash in case some undefined status is returned for reasons unknown.

Note: For kwargs added by the decorators, see *the decorator documentation*.

lbd_backend.LBD_REST_locationdata.views.**api**(*request*, **args*, ***kwargs*)
This view returns the installed open data sources as JSON.

Supported HTTP methods:

•GET

Returns HTTP response.

lbd_backend.LBD_REST_locationdata.views.**collection**(*request*, **args*, ***kwargs*)
REST main collection request handler.

Supported HTTP methods:

•GET

•DELETE

•PUT

•POST

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

In addition to the kwargs added by the decorators, this view uses the following:

•collection (String)

–Location data collection name

Supported URL parameter:

•mini (True or False): Return minimum amount of data (response must still be valid GeoJSON

lbd_backend.LBD_REST_locationdata.views.**collection_inarea**(*request*, **args*, ***kwargs*)
REST subcollection "inarea" request handler. Handles objects inside a rectangular area.

Supported HTTP methods:

•GET

•DELETE

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

In addition to the kwargs added by the decorators, this view uses the following:

•collection (String)

–Location data collection name

Supported URL parameter:

•xbottomleft (Float): The x-coordinate of the bottom left corner of the area

•ybottomleft (Float): The y-coordinate of the bottom left corner of the area

•xtopright (Float): The x-coordinate of the top right corner of the area

•ytopright (Float): The y-coordinate of the top right corner of the area

•mini (True or False): Return minimum amount of data (response must still be valid GeoJSON

lbd_backend.LBD_REST_locationdata.views.**collection_near**(*request*, **args*, ***kwargs*)

REST subcollection "near" request handler. Handles objects in certain range of given coordinates

Supported HTTP methods:

•GET

•DELETE

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

In addition to the kwargs added by the decorators, this view uses the following:

•collection (String)

–Location data collection name

Supported URL parameter:

•latitude (Float): the latitude of the center **REQUIRED**

•longitude (Float): the longitude of the center **REQUIRED**

•range (Float): the radius of the area

•mini (True or False): Return minimum amount of data (response must still be valid GeoJSON

lbd_backend.LBD_REST_locationdata.views.**search_from_rest**(*request*, **args*, ***kwargs*)

This view searches for the given search phrase from the database. Currently only search from id field is supported. For json format, see *Search*.

Supported HTTP methods:

•POST

Parameters

- **request** – Request object
- **args** – Arguments
- **kwargs** – Keyword arguments

Returns HTTP response

lbd_backend.LBD_REST_locationdata.views.**single_resource**(*request*, **args*, ***kwargs*)

REST single resource (in certain collection) request handler.

Supported HTTP methods:

•GET

•DELETE

•PUT

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

In addition to the kwargs added by the decorators, this view uses the following:

•collection (String)

–Location data collection name

•resource (String)

–Resource id

Returns HTTP response. Possible statuses are listed in module documentation

6.2.2 Model containing the metadata database structure

class lbd_backend.LBD_REST_locationdata.models.**MetaData**(**args*, ***kwargs*)
Fields:

•status: StringField **REQUIRED**

•modified: IntField **REQUIRED**

•modifier: IntField **REQUIRED**

•info: StringField **REQUIRED**

Status is a string describing the status of the object. It is always required if metadata for the object is defined.

Modified is a timestamp (seconds from epoch) and is generated automatically by the system. Always required.

Modifier is the id of the user that modified the metadata item. Always required, inserted by the system.

Info is ... infofield?

New fields can be dynamically added into this model.

class lbd_backend.LBD_REST_locationdata.models.**MetaDocument**(**args*, ***values*)
Fields

•feature_id: StringField **REQUIRED UNIQUE**

•collection: StringField **REQUIRED**

•meta_data: EmbeddeDocumentField(MetaData) **REQUIRED**

Feature_id is a string that combines the metadata to an object. Simulates a foreign key.

Collection_id is a string that tells the collection where the metadata belongs.

Meta_data is an embedded document.

New fields can be dynamically added into this model.

6.3 lbd_backend.LBD_REST_messagedata Message data REST

6.3.1 View for handling messages

lbd_backend.LBD_REST_messagedata.views.**mark_as_read**(*request*, **args*, ***kwargs*)

View for marking a message read.

Supported HTTP methods:

•GET

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

The method uses the following kwargs:

•message (Integer)

•lbduser (User)

Message specifies the message id. Required.

Returns HTTP response. Possible statuses are listed in module documentation

lbd_backend.LBD_REST_messagedata.views.**msg_general**(*request*, **args*, ***kwargs*)

Handles all message requests (both to single and multiple messages).

Supported HTTP methods:

•GET

•DELETE

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

The method uses the following kwargs:

•category (String)

•message (Integer)

•lbduser (User)

Category specifies the message category. This argument is used only if it is specified in the url. Category is equivalent to locationdata collection. If this argument is used, it is expected that a locationdata collection with the same name exists and is "installed".

Message specifies the message id. Used only if specified in the url.

Returns HTTP response. Possible statuses are listed in module documentation

lbd_backend.LBD_REST_messagedata.views.**msg_send**(*request*, **args*, ***kwargs*)
View for sending messages.

Supported HTTP methods:

•POST

Parameters

- **request** – Request object
- **args** – arguments
- **kwargs** – Dictionary (keyword arguments). Known kwargs listed below.

The method uses the following kwargs:

•lbduser (User)

Returns HTTP response. Possible statuses are listed in module documentation

6.3.2 Model for message

class lbd_backend.LBD_REST_messagedata.models.**Attachment**(**args*, ***kwargs*)
Fields:

•category: StringField **REQUIRED**

•aid: StringField **REQUIRED**

category is the name of the locationdata collection to which the attachment refers.

aid is the id (not mongodb id, but the back-end's id) of the object to which the attachment refers ("foreignkey").

class lbd_backend.LBD_REST_messagedata.models.**Message**(**args*, ***values*)
Fields:

•mid: SequenceField **AUTOMATIC**

•category: StringField

•sender: EmailField **REQUIRED**

•recipient: EmailField **REQUIRED**

•attachments: ListField(EmbeddedDocumentField(Attachment))

•topic: StringField **REQUIRED**

•message: StringField **REQUIRED**

•messageread: BooleanField **REQUIRED**

•timestamp: IntField **REQUIRED**

Mid is the id of the message. Generated automatically.

Category is the name of the locationdata collection to which the message refers.

Sender is the email address of the sender.

Recipient is the email address of the recipient.

Attachment is a list of Attachment objects.

Topic is the topic of the message.

Message is the message content.

Messageread tells if the message has been read or not. (True or False) (False by default)

Timestamp tells when the message was sent. Timestamp is in seconds from Unix Epoch on January 1st, 1970 at UTC.

CHAPTER

SEVEN

CODE DOCUMENTATION: HANDLERS

7.1 RESThandlers.HandlerInterface REST Handler Interface

exception RESThandlers.HandlerInterface.Exceptions.**CollectionNotInstalled**
Bases: exceptions.Exception

exception RESThandlers.HandlerInterface.Exceptions.**GenericDBError**
Bases: exceptions.Exception

exception RESThandlers.HandlerInterface.Exceptions.**MultipleObjectsFound**
Bases: exceptions.Exception

exception RESThandlers.HandlerInterface.Exceptions.**ObjectNotFound**
Bases: exceptions.Exception

class RESThandlers.HandlerInterface.HandlerBaseClass.**HandlerBase**
Bases: object

delete_all()

delete_item_by_id(*iid*)

delete_near(*latitude*, *longitude*, *nrange*)

get_all(*mini=False*)

get_by_id(*iid*)

get_field_names()

get_item_count()

get_near(*longitude*, *latitude*, *nrange*)

get_within_rectangle(*xtop_right*, *ytop_right*, *xbottom_left*, *ybottom_left*, *mini=False*)

handler_id
staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C: def f(arg1, arg2, ...): ... f = staticmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

insert_to_db(*jsonitem*)

search(*phrase*, *field*)

update_db()

7.2 RESThandlers.HandlerInterface.Factory Handler Factory

class RESThandlers.HandlerInterface.Factory.**HandlerFactory**(*collection*)
Bases: object

create()
Creates and return a handler object.

Returns Handler object

static **get_installed**()
Static

This method returns the installed open data services as dictionary where "name" is the name of the service (used when creating a new handler object) and "fields" tells what elements the service provides. :return: dictionary

7.3 RESThandlers.Streetlight Streetlight REST handler

class RESThandlers.Streetlight.Handler.**StreetlightHandler**
Bases: RESThandlers.HandlerInterface.HandlerBaseClass.HandlerBase

delete_all()

delete_near(*latitude*, *longitude*, *nrange*)

get_all(*mini=True*)

get_by_id(*iid*)

get_field_names()

get_item_count()

get_near(*longitude*, *latitude*, *nrange=0.001*, *mini=False*)

get_within_rectangle(*xtop_right*, *ytop_right*, *xbottom_left*, *ybottom_left*, *mini=False*)

handler_id
staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C: def f(arg1, arg2, ...): ... f = staticmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

search(*regex*, *limit*, *field=None*)

update_db()

7.4 RESThandlers.Playgrounds Playgrounds REST handler

class RESThandlers.Playgrounds.Handler.**PlaygroundHandler**

Bases: RESThandlers.HandlerInterface.HandlerBaseClass.HandlerBase

delete_all()

delete_near(*latitude*, *longitude*, *nrange*)

get_all(*mini=True*)

get_by_id(*iid*)

get_field_names()

get_item_count()

get_near(*longitude*, *latitude*, *nrange=0.001*, *mini=False*)

get_within_rectangle(*xtop_right*, *ytop_right*, *xbottom_left*, *ybottom_left*, *mini=False*)

handler_id

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C: def f(arg1, arg2, ...): ... f = staticmethod(f)

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

search(*regex*, *limit*, *field=None*)

update_db()

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

h

l

m

r

A

C

D

G

H

I

L

M

O

P

R

S

U