Python Overpass API Documentation

Release 0.4

PhiBo

Apr 07, 2017

Contents

Contents:

CHAPTER 1

Introduction

Requirements

Supported Python versions:

- Python 2.7
- Python > 3.2
- PyPy

Installation

As a Python egg

You can install the most recent version using `pip`

```
$ pip install overpy
```

From a tarball release

Download the most recent tarball from github, unpack it and run the following command on the command-line.

```
$ python setup.py install
```

Install the development version

Install git and run the following commands on the command-line.

```
$ git clone https://github.com/DinoTools/python-overpy.git
$ cd python-overpy
$ python setup.py install
```

Usage

It is recommended to have a look at the documentation of the Overpass API before using OverPy. For more examples have a look at the *examples page* or in the examples directory.

```
import overpy

api = overpy.Overpass()

# fetch all ways and nodes
result = api.query("""
    way(50.746,7.154,50.748,7.157) ["highway"];
    (._;>;);
    out body;
    """)

for way in result.ways:
    print("Name: %s" % way.tags.get("name", "n/a"))
    print("  Highway: %s" % way.tags.get("highway", "n/a"))
    print("  Nodes:")
    for node in way.nodes:
        print("    Lat: %f, Lon: %f" % (node.lat, node.lon))
```

CHAPTER 2

Examples

Basic example

Lets start with an example from the Overpass API documentation.

Query String:

```
node(50.745,7.17,50.75,7.18);
out;
```

Use OverPy:

```
>>> import overpy
>>> api = overpy.Overpass()
>>> result = api.query("node(50.745,7.17,50.75,7.18);out;")
>>> len(result.nodes)
1984
>>> len(result.ways)
0
>>> len(result.relations)
0
>>> node = result.nodes[2]
>>> node.id
100792806
>>> node.tags
{}
```

Line 1: Import the required Python module

Line 2: Create a new instance of the Overpass() class. This instance is used to query the Overpass API.

Line 3: Use the Query-String from above to query the Overpass API service.

Line 4,5: Get the number of nodes in the result set.

Line 6-9: Get the number of ways and relations available in the result set.

Line 10-14: Get the third node from the list. Display the ID and the tags of this node.

Use Overpass QL or Overpass XML

Queries are passed directly to the Overpass API service without any modification. So it is possible to use Overpass QL and Overpass XML.

Overpass QL

Query:

```
node["name"="Gielgen"];
out body;
```

Use OverPy:

```
>>> import overpy
>>> api = overpy.Overpass()
>>> result = api.query("""node["name"="Gielgen"];out body;""")
>>> len(result.nodes)
6
>>> len(result.ways)
0
>>> len(result.relations)
0
```

Overpass XML

Query:

```
<osm-script>
  <query type="node">
    <has-kv k="name" v="Gielgen"/>
  </query>
  <print/>
</osm-script>
```

Use OverPy:

```
>>> import overpy
>>> api = overpy.Overpass()
>>> result = api.query("""<osm-script>
...   <query type="node">
...     <has-kv k="name" v="Gielgen"/>
...   </query>
...   <print/>
... </osm-script>""")
>>> len(result.nodes)
6
>>> len(result.ways)
0
>>> len(result.relations)
0
```

Parse JSON or XML responses

On a request OverPy detects the content type from the response.

JSON response

Query String:

```
[out:json];
node(50.745,7.17,50.75,7.18);
out;
```

Use OverPy:

```
>>> import overpy
>>> api = overpy.Overpass()
>>> result = api.query("[out:json];node(50.745,7.17,50.75,7.18);out;")
>>> len(result.nodes)
1984
>>> len(result.ways)
0
>>> len(result.relations)
0
```

XML response

Query String:

```
[out:xml];
node(50.745,7.17,50.75,7.18);
out;
```

Use OverPy:

```
>>> import overpy
>>> api = overpy.Overpass()
>>> result = api.query("[out:xml];node(50.745,7.17,50.75,7.18);out;")
>>> len(result.nodes)
1984
>>> len(result.ways)
0
>>> len(result.relations)
0
```

Ways

Get all nodes of a way

In this example the Overpass API will only return the Way elements with the name “Gielgenstraße”. But there will be no Node elements in the result set.

OverPy provides a way to resolve missing nodes.

Query String:

```
way
["name"="Gielgenstraße"]
(50.7,7.1,50.8,7.25);
out;
```

Use OverPy:

```
>>> import overpy
>>> api = overpy.Overpass()
>>> result = api.query("""way["name"="Gielgenstraße"](50.7,7.1,50.8,7.25);out;""")
>>> len(result.nodes)
0
>>> len(result.ways)
4
>>> way = result.ways[0]
>>> way.nodes
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    [...]
    raise exception.DataIncomplete("Resolve missing nodes is disabled")
overpy.exception.DataIncomplete: ('Data incomplete try to improve the query to␣
↪resolve the missing data', 'Resolve missing nodes is disabled')
>>> way.get_nodes()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    [...]
    raise exception.DataIncomplete("Resolve missing nodes is disabled")
overpy.exception.DataIncomplete: ('Data incomplete try to improve the query to␣
↪resolve the missing data', 'Resolve missing nodes is disabled')
>>> nodes = way.get_nodes(resolve_missing=True)
>>> len(nodes)
13
>>> len(result.nodes)
13
>>> len(way.nodes)
13
```

Line 1-3: Send a query to the Overpass API service.

Line 4-6: There are 4 Way elements and 0 Node elements in the result set.

Line 7: Get the first way.

Line 8-19: Use *overpy.Way.nodes* class attribute and the *overpy.Way.get_nodes()* function to get the nodes for the way. Both raise an exception because the nodes are not in the result set and auto resolving missing nodes is disabled.

Line 20-21: Use the *overpy.Way.get_nodes()* function and let OverPy try to resolve the missing nodes. The function will return all Node elements connected with the Way element.

Line 22-25: The resolved nodes have been added to the result set and are available to be used again later.

CHAPTER 3

API Reference

Overpass API

class overpy.**Overpass**(*read_chunk_size=None*, *url=None*, *xml_parser=2*)

Class to access the Overpass API

parse_json(*data*, *encoding='utf-8'*)

Parse raw response from Overpass service.

Parameters

- **data** (*String or Bytes*) – Raw JSON Data
- **encoding** (*String*) – Encoding to decode byte string

Returns Result object

Return type *overpy.Result*

parse_xml(*data*, *encoding='utf-8'*, *parser=None*)

Parameters

- **data** (*String or Bytes*) – Raw XML Data
- **encoding** (*String*) – Encoding to decode byte string

Returns Result object

Return type *overpy.Result*

query(*query*)

Query the Overpass API

Parameters **query** (*String|Bytes*) – The query string in Overpass QL

Returns The parsed result

Return type *overpy.Result*

Result

class overpy.**Result**(*elements=None*, *api=None*)
Class to handle the result.

append(*element*)
Append a new element to the result.

Parameters **element** (overpy.Element) – The element to append

areas
Alias for get_elements() but filter the result by Area

Parameters **area_id** (*Integer*) – The Id of the area

Returns List of elements

expand(*other*)
Add all elements from an other result to the list of elements of this result object.

It is used by the auto resolve feature.

Parameters **other** (overpy.Result) – Expand the result with the elements from this result.

Raises **ValueError** – If provided parameter is not instance of *overpy.Result*

classmethod **from_json**(*data*, *api=None*)
Create a new instance and load data from json object.

Parameters

- **data** (*Dict*) – JSON data returned by the Overpass API
- **api** (overpy.Overpass) –

Returns New instance of Result object

Return type *overpy.Result*

classmethod **from_xml**(*data*, *api=None*, *parser=2*)
Create a new instance and load data from xml object.

Parameters

- **data** (*xml.etree.ElementTree.Element*) – Root element
- **api** (Overpass) –
- **parser** (*Integer*) – Specify the parser to use(DOM or SAX)

Returns New instance of Result object

Return type *Result*

get_area(*area_id*, *resolve_missing=False*)
Get an area by its ID.

Parameters

- **area_id** (*Integer*) – The area ID
- **resolve_missing** – Query the Overpass API if the area is missing in the result set.

Returns The area

Return type *overpy.Area*

Raises

- **overpy.exception.DataIncomplete** – The requested way is not available in the result cache.
- **overpy.exception.DataIncomplete** – If resolve_missing is True and the area can't be resolved.

get_areas(*area_id=None*, ***kwargs*)
Alias for get_elements() but filter the result by Area

Parameters **area_id** (*Integer*) – The Id of the area

Returns List of elements

get_elements(*filter_cls*, *elem_id=None*)
Get a list of elements from the result and filter the element type by a class.

Parameters

- **filter_cls** –
- **elem_id** (*Integer*) – ID of the object

Returns List of available elements

Return type List

get_ids(*filter_cls*)

Parameters **filter_cls** –

Returns

get_node(*node_id*, *resolve_missing=False*)
Get a node by its ID.

Parameters

- **node_id** (*Integer*) – The node ID
- **resolve_missing** – Query the Overpass API if the node is missing in the result set.

Returns The node

Return type *overpy.Node*

Raises

- **overpy.exception.DataIncomplete** – At least one referenced node is not available in the result cache.
- **overpy.exception.DataIncomplete** – If resolve_missing is True and at least one node can't be resolved.

get_nodes(*node_id=None*, ***kwargs*)
Alias for get_elements() but filter the result by Node()

Parameters **node_id** (*Integer*) – The Id of the node

Returns List of elements

get_relation(*rel_id*, *resolve_missing=False*)
Get a relation by its ID.

Parameters

- **rel_id** (*Integer*) – The relation ID
- **resolve_missing** – Query the Overpass API if the relation is missing in the result set.

Returns The relation

Return type *overpy.Relation*

Raises

- **overpy.exception.DataIncomplete** – The requested relation is not available in the result cache.
- **overpy.exception.DataIncomplete** – If resolve_missing is True and the relation can't be resolved.

get_relations(*rel_id=None*, ***kwargs*)
Alias for get_elements() but filter the result by Relation

Parameters **rel_id** (*Integer*) – Id of the relation

Returns List of elements

get_way(*way_id*, *resolve_missing=False*)
Get a way by its ID.

Parameters

- **way_id** (*Integer*) – The way ID
- **resolve_missing** – Query the Overpass API if the way is missing in the result set.

Returns The way

Return type *overpy.Way*

Raises

- **overpy.exception.DataIncomplete** – The requested way is not available in the result cache.
- **overpy.exception.DataIncomplete** – If resolve_missing is True and the way can't be resolved.

get_ways(*way_id=None*, ***kwargs*)
Alias for get_elements() but filter the result by Way

Parameters **way_id** (*Integer*) – The Id of the way

Returns List of elements

nodes
Alias for get_elements() but filter the result by Node()

Parameters **node_id** (*Integer*) – The Id of the node

Returns List of elements

relations
Alias for get_elements() but filter the result by Relation

Parameters **rel_id** (*Integer*) – Id of the relation

Returns List of elements

ways
Alias for get_elements() but filter the result by Way

Parameters **way_id** (*Integer*) – The Id of the way

Returns List of elements

Elements

class overpy.**Element**(*attributes=None*, *result=None*, *tags=None*)
Base element

classmethod get_center_from_json(*data*)
Get center information from json data

Parameters **data** – json data

Returns tuple with two elements: lat and lon

Return type tuple

class overpy.**Area**(*area_id=None*, ***kwargs*)
Class to represent an element of type area

classmethod from_json(*data*, *result=None*)
Create new Area element from JSON data

Parameters

- **data** (*Dict*) – Element data from JSON
- **result** (overpy.Result) – The result this element belongs to

Returns New instance of Way

Return type *overpy.Area*

Raises ***overpy.exception.ElementDataWrongType*** – If type value of the passed JSON data does not match.

classmethod from_xml(*child*, *result=None*)
Create new way element from XML data

Parameters

- **child** (*xml.etree.ElementTree.Element*) – XML node to be parsed
- **result** (overpy.Result) – The result this node belongs to

Returns New Way oject

Return type *overpy.Way*

Raises

- ***overpy.exception.ElementDataWrongType*** – If name of the xml child node doesn't match
- **ValueError** – If the ref attribute of the xml node is not provided
- **ValueError** – If a tag doesn't have a name

id = None
The id of the way

class overpy.**Node**(*node_id=None*, *lat=None*, *lon=None*, ***kwargs*)
Class to represent an element of type node

classmethod from_json(*data*, *result=None*)
Create new Node element from JSON data

Parameters

- **data** (*Dict*) – Element data from JSON

- **result** (overpy.Result) – The result this element belongs to

Returns New instance of Node

Return type *overpy.Node*

Raises ***overpy.exception.ElementDataWrongType*** – If type value of the passed JSON data does not match.

classmethod from_xml(*child*, *result=None*)
Create new way element from XML data

Parameters

- **child** (*xml.etree.ElementTree.Element*) – XML node to be parsed
- **result** (overpy.Result) – The result this node belongs to

Returns New Way oject

Return type *overpy.Node*

Raises

- ***overpy.exception.ElementDataWrongType*** – If name of the xml child node doesn't match
- **ValueError** – If a tag doesn't have a name

class overpy.**Relation**(*rel_id=None*, *center_lat=None*, *center_lon=None*, *members=None*, ***kwargs*)
Class to represent an element of type relation

center_lat = None
The lat/lon of the center of the way (optional depending on query)

classmethod from_json(*data*, *result=None*)
Create new Relation element from JSON data

Parameters

- **data** (*Dict*) – Element data from JSON
- **result** (overpy.Result) – The result this element belongs to

Returns New instance of Relation

Return type *overpy.Relation*

Raises ***overpy.exception.ElementDataWrongType*** – If type value of the passed JSON data does not match.

classmethod from_xml(*child*, *result=None*)
Create new way element from XML data

Parameters

- **child** (*xml.etree.ElementTree.Element*) – XML node to be parsed
- **result** (overpy.Result) – The result this node belongs to

Returns New Way oject

Return type *overpy.Relation*

Raises

- ***overpy.exception.ElementDataWrongType*** – If name of the xml child node doesn't match

- **ValueError** – If a tag doesn't have a name

class overpy.**Way**(*way_id=None*, *center_lat=None*, *center_lon=None*, *node_ids=None*, ***kwargs*)
Class to represent an element of type way

center_lat = None
The lat/lon of the center of the way (optional depending on query)

classmethod from_json(*data*, *result=None*)
Create new Way element from JSON data

Parameters

- **data** (*Dict*) – Element data from JSON
- **result** (overpy.Result) – The result this element belongs to

Returns New instance of Way

Return type *overpy.Way*

Raises ***overpy.exception.ElementDataWrongType*** – If type value of the passed JSON data does not match.

classmethod from_xml(*child*, *result=None*)
Create new way element from XML data

Parameters

- **child** (*xml.etree.ElementTree.Element*) – XML node to be parsed
- **result** (overpy.Result) – The result this node belongs to

Returns New Way oject

Return type *overpy.Way*

Raises

- ***overpy.exception.ElementDataWrongType*** – If name of the xml child node doesn't match
- **ValueError** – If the ref attribute of the xml node is not provided
- **ValueError** – If a tag doesn't have a name

get_nodes(*resolve_missing=False*)
Get the nodes defining the geometry of the way

Parameters **resolve_missing** (*Boolean*) – Try to resolve missing nodes.

Returns List of nodes

Return type List of overpy.Node

Raises

- ***overpy.exception.DataIncomplete*** – At least one referenced node is not available in the result cache.
- ***overpy.exception.DataIncomplete*** – If resolve_missing is True and at least one node can't be resolved.

id = None
The id of the way

nodes
List of nodes associated with the way.

Relation Members

class overpy.**RelationMember**(*attributes=None*, *geometry=None*, *ref=None*, *role=None*, *result=None*)
Base class to represent a member of a relation.

classmethod **from_json**(*data*, *result=None*)
Create new RelationMember element from JSON data

Parameters

- **child** (*Dict*) – Element data from JSON
- **result** (overpy.Result) – The result this element belongs to

Returns New instance of RelationMember

Return type *overpy.RelationMember*

Raises ***overpy.exception.ElementDataWrongType*** – If type value of the passed JSON data does not match.

classmethod **from_xml**(*child*, *result=None*)
Create new RelationMember from XML data

Parameters

- **child** (*xml.etree.ElementTree.Element*) – XML node to be parsed
- **result** (overpy.Result) – The result this element belongs to

Returns New relation member oject

Return type *overpy.RelationMember*

Raises ***overpy.exception.ElementDataWrongType*** – If name of the xml child node doesn't match

class overpy.**RelationArea**(*attributes=None*, *geometry=None*, *ref=None*, *role=None*, *result=None*)

class overpy.**RelationNode**(*attributes=None*, *geometry=None*, *ref=None*, *role=None*, *result=None*)

class overpy.**RelationWay**(*attributes=None*, *geometry=None*, *ref=None*, *role=None*, *result=None*)

Exceptions

exception overpy.exception.**DataIncomplete**(**args*, ***kwargs*)
Raised if the requested data isn't available in the result. Try to improve the query or to resolve the missing data.

exception overpy.exception.**ElementDataWrongType**(*type_expected*, *type_provided=None*)
Raised if the provided element does not match the expected type.

Parameters

- **type_expected** (*String*) – The expected element type
- **type_provided** (*String|None*) – The provided element type

exception overpy.exception.**OverPyException**
OverPy base exception

exception overpy.exception.**OverpassBadRequest**(*query*, *msgs=None*)
Raised if the Overpass API service returns a syntax error.

Parameters

- **query** (*Bytes*) – The encoded query how it was send to the server
- **msgs** (*List*) – List of error messages

exception overpy.exception.**OverpassGatewayTimeout**
Raised if load of the Overpass API service is too high and it can't handle the request.

exception overpy.exception.**OverpassTooManyRequests**
Raised if the Overpass API service returns a 429 status code.

exception overpy.exception.**OverpassUnknownContentType**(*content_type*)
Raised if the reported content type isn't handled by OverPy.

Parameters **content_type** (*None or String*) – The reported content type

exception overpy.exception.**OverpassUnknownHTTPStatusCode**(*code*)
Raised if the returned HTTP status code isn't handled by OverPy.

Parameters **code** (*Integer*) – The HTTP status code

Helper

overpy.helper.**get_intersection**(*street1*, *street2*, *areacode*, *api=None*)
Retrieve intersection of two streets in a given bounding area

Parameters

- **api** (overpy.Overpass) – First street of intersection
- **street1** (*String*) – Name of first street of intersection
- **street2** (*String*) – Name of second street of intersection
- **areacode** (*String*) – The OSM id of the bounding area

Returns List of intersections

Raises ***overpy.exception.OverPyException*** – If something bad happens.

overpy.helper.**get_street**(*street*, *areacode*, *api=None*)
Retrieve streets in a given bounding area

Parameters

- **api** (overpy.Overpass) – First street of intersection
- **street** (*String*) – Name of street
- **areacode** (*String*) – The OSM id of the bounding area

Returns Parsed result

Raises ***overpy.exception.OverPyException*** – If something bad happens.

CHAPTER 4

Contributing

First of all, thank you for your interest in contributing to OverPy!

Filing bug reports

Bug reports are very welcome. Please fill them on the GitHub issue tracker. Good bug reports come with extensive descriptions of the error and how to reproduce it.

Patches

All patches to OverPy should be submitted in the form of pull requests to the main OverPy repository, DinoTools/python-overpy. These pull requests should satisfy the following properties:

Code

- The pull request should focus on one particular improvement to OverPy.
- Create different pull requests for unrelated features or bugfixes.
- Python code should follow PEP 8, especially in the "do what code around you does" sense.

Documentation

When introducing new functionality, please remember to write documentation.

Tests

It is recommended to add tests for new code you add.

Review

Finally, pull requests must be reviewed before merging. Everyone can perform reviews; this is a very valuable way to contribute, and is highly encouraged.

CHAPTER 5

Changelog

0.x (master)

Note: This version is not yet released and is under development.

0.4 (2016-12-08)

- Add SAX parser
- Add option to choose DOM or SAX parser
- Fix issues with CI builds with Python 3.2
- Add Python 3.5 to CI builds
- Fix issues (Thanks to all contributors)
- Add property for default API URL
- Add examples
- Build Fixes
- GitHub templates
- Parse center information
- Parse geometry information
- Support Areas

0.3.1 (2015-04-30)

- Improve example

0.3.0 (2015-04-30)

- Improve internal data handling (Dominik)
- Add helper functions (Morris Jobke)

0.2.0 (2014-12-27)

- Added support for xml response data
- Added support for exceptions
- Added tests with 100% code coverage
- Removed Python 2.6 support
- Added more examples to the documentation

0.1.0 (2014-12-14)

Proof of concept

- Initial release.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

W