Python Overpass API Documentation

Release 0.6

PhiBo

Aug 16, 2023

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 Requirements

Supported Python versions:

- Python 3.6+
- PyPy3

1.2 Installation

1.2.1 As a Python egg

You can install the most recent version using `pip`

```
$ pip install overpy
```

1.2.2 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
```

1.2.3 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
```

1.3 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

TWO

EXAMPLES

2.1 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.

2.2 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.

2.2.1 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
```

2.2.2 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
```

2.3 Parse JSON or XML responses

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

2.3.1 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
```

2.3.2 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
```

2.4 Ways

2.4.1 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

THREE

FREQUENTLY ASKED QUESTIONS

3.1 429 Too Many Requests

If too many requests are send from the same IP address the server blocks some requests to avoid that a user uses up all resources. For more information have a look at the Overpass API documentation.

OverPy tries to fetch missing information automatically. To limit the number of requests you should try to fetch all required information/data(relations, ways, nodes, tags, ...) with the initial query.

CHAPTER

FOUR

API REFERENCE

4.1 Overpass API

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

Class to access the Overpass API

Variables

- **default_max_retry_count** – Global max number of retries (Default: 0)
- **default_read_chunk_size** – Max size of each chunk read from the server response
- **default_retry_timeout** – Global time to wait between tries (Default: 1.0s)
- **default_url** – Default URL of the Overpass server

Parameters

- **read_chunk_size** (Optional[int]) –
- **url** (Optional[str]) –
- **xml_parser** (int) –
- **max_retry_count** (Optional[int]) –
- **retry_timeout** (Optional[float]) –

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

Parse raw response from Overpass service.

Parameters

- **data** (Union[bytes, str]) – Raw JSON Data
- **encoding** (str) – Encoding to decode byte string

Return type
Result

Returns
Result object

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

Parameters

- **data** (Union[bytes, str]) – Raw XML Data
- **encoding** (str) – Encoding to decode byte string

- **parser** (Optional[int]) – The XML parser to use

Returns
Result object

query(*query*)

Query the Overpass API

Parameters
query (Union[bytes, str]) – The query string in Overpass QL

Return type
Result

Returns
The parsed result

4.2 Result

class overpy.**Result**(*elements=None*, *api=None*)

Class to handle the result.

Parameters

- **elements** (Optional[List[Union[*Area*, *Node*, *Relation*, *Way*]]]) –
- **api** (Optional[*Overpass*]) –

append(*element*)

Append a new element to the result.

Parameters
element (Union[*Area*, *Node*, *Relation*, *Way*]) – The element to append

property areas: List[*Area*]

Alias for get_elements() but filter the result by Area

Parameters
area_id (Optional[int]) – 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 (*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** (Optional[*Overpass*]) –

Return type
Result

Returns
New instance of Result object

classmethod from_xml(*data*, *api=None*, *parser=None*)

Create a new instance and load data from xml data or object.

Note: If parser is set to None, the functions tries to find the best parse. By default the SAX parser is chosen if a string is provided as data. The parser is set to DOM if an xml.etree.ElementTree.Element is provided as data value.

Parameters

- **data** (Union[str, Element]) – Root element
- **api** (Optional[*Overpass*]) – The instance to query additional information if required.
- **parser** (Optional[int]) – Specify the parser to use(DOM or SAX)(Default: None = autodetect, defaults to SAX)

Return type
Result

Returns
New instance of Result object

get_area(*area_id*, *resolve_missing=False*)

Get an area by its ID.

Parameters

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

Return type
Area

Returns
The 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*)

Alias for get_elements() but filter the result by Area

Parameters
area_id (Optional[int]) – The Id of the area

Return type
List[*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** (Type[TypeVar(ElementTypeVar, bound= Element)]) –
- **elem_id** (Optional[int]) – ID of the object

Return type
List[TypeVar(ElementTypeVar, bound= Element)]

Returns
List of available elements

get_ids(*filter_cls*)

Get all Element IDs

Parameters
filter_cls (Type[Union[*Area*, *Node*, *Relation*, *Way*]]) – Only IDs of elements with this type

Return type
List[int]

Returns
List of IDs

get_node(*node_id*, *resolve_missing=False*)

Get a node by its ID.

Parameters

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

Return type
Node

Returns
The 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*)

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

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

Return type
List[*Node*]

Returns
List of elements

get_relation(*rel_id*, *resolve_missing=False*)

Get a relation by its ID.

Parameters

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

Return type
Relation

Returns
The 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*)

Alias for get_elements() but filter the result by Relation

Parameters
rel_id (Optional[int]) – Id of the relation

Return type
List[*Relation*]

Returns
List of elements

get_way(*way_id*, *resolve_missing=False*)

Get a way by its ID.

Parameters

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

Return type
Way

Returns
The 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*)

Alias for get_elements() but filter the result by Way

Parameters
way_id (Optional[int]) – The Id of the way

Return type
List[*Way*]

Returns
List of elements

property nodes: List[*Node*]

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

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

Returns
List of elements

property relations: List[*Relation*]

Alias for get_elements() but filter the result by Relation

Parameters
rel_id (Optional[int]) – Id of the relation

Returns
List of elements

property ways: List[*Way*]

Alias for get_elements() but filter the result by Way

Parameters
way_id (Optional[int]) – The Id of the way

Returns
List of elements

4.3 Elements

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

Base element

Parameters

- **attributes** (Optional[dict]) –
- **result** (Optional[*Result*]) –
- **tags** (Optional[Dict]) –

classmethod from_json(*data*, *result=None*)

Create new Element() from json data :type data: dict :param data: :type result: Optional[*Result*] :param result: :rtype: TypeVar(ElementTypeVar, bound= Element) :return:

classmethod from_xml(*child*, *result=None*)

Create new Element() element from XML data

Parameters

- **child** (Element) –
- **result** (Optional[*Result*]) –

Return type
TypeVar(ElementTypeVar, bound= Element)

classmethod get_center_from_json(*data*)

Get center information from json data

Parameters
data (dict) – json data

Return type
Tuple[Decimal, Decimal]

Returns
tuple with two elements: lat and lon

class overpy.**Area**(*area_id=None*, ***kwargs*)

Class to represent an element of type area

Parameters
area_id (Optional[int]) –

classmethod from_json(*data*, *result=None*)

Create new Area element from JSON data

Parameters

- **data** (dict) – Element data from JSON
- **result** (Optional[*Result*]) – The result this element belongs to

Return type
Area

Returns
New instance of 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** (Element) – XML node to be parsed
- **result** (Optional[*Result*]) – The result this node belongs to

Return type
Area

Returns
New Way oject

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: int

The id of the way

class overpy.**Node**(*node_id=None*, *lat=None*, *lon=None*, ***kwargs*)

Class to represent an element of type node

Parameters

- **node_id** (Optional[int]) –
- **lat** (Union[Decimal, float, None]) –
- **lon** (Union[Decimal, float, None]) –

classmethod from_json(*data*, *result=None*)

Create new Node element from JSON data

Parameters

- **data** (dict) – Element data from JSON
- **result** (Optional[*Result*]) – The result this element belongs to

Return type
Node

Returns
New instance of 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** (Element) – XML node to be parsed
- **result** (Optional[*Result*]) – The result this node belongs to

Return type
Node

Returns
New Way oject

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

Parameters

- **rel_id** (Optional[int]) –
- **center_lat** (Union[Decimal, float, None]) –

- **center_lon** (Union[Decimal, float, None]) –
- **members** (Optional[List[*RelationMember*]]) –

center_lat

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** (Optional[*Result*]) – The result this element belongs to

Return type
Relation

Returns
New instance of 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** (Element) – XML node to be parsed
- **result** (Optional[*Result*]) – The result this node belongs to

Return type
Relation

Returns
New Way oject

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

Parameters

- **way_id** (Optional[int]) –
- **center_lat** (Union[Decimal, float, None]) –
- **center_lon** (Union[Decimal, float, None]) –
- **node_ids** (Union[List[int], Tuple[int], None]) –

center_lat

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** (Optional[*Result*]) – The result this element belongs to

Return type
Way

Returns
New instance of 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** (Element) – XML node to be parsed
- **result** (Optional[*Result*]) – The result this node belongs to

Return type
Way

Returns
New Way oject

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 (bool) – Try to resolve missing nodes.

Return type
List[*Node*]

Returns
List of nodes

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: int
The id of the way

property nodes: List[*Node*]
List of nodes associated with the way.

4.4 Relation Members

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

Parameters

- **attributes** (Optional[dict]) –
- **geometry** (Optional[List[RelationWayGeometryValue]]) –
- **ref** (Optional[int]) –
- **role** (Optional[str]) –
- **result** (Optional[*Result*]) –

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

Parameters

- **data** (dict) – Element data from JSON
- **result** (Optional[*Result*]) – The result this element belongs to

Return type
RelationMember

Returns
New instance of 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** (Element) – XML node to be parsed
- **result** (Optional[*Result*]) – The result this element belongs to

Return type
RelationMember

Returns
New relation member oject

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*)

Parameters

- **attributes** (Optional[dict]) –
- **geometry** (Optional[List[RelationWayGeometryValue]]) –
- **ref** (Optional[int]) –
- **role** (Optional[str]) –
- **result** (Optional[*Result*]) –

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

Parameters

- **attributes** (Optional[dict]) –
- **geometry** (Optional[List[RelationWayGeometryValue]]) –
- **ref** (Optional[int]) –
- **role** (Optional[str]) –
- **result** (Optional[*Result*]) –

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

Parameters

- **attributes** (Optional[dict]) –
- **geometry** (Optional[List[RelationWayGeometryValue]]) –
- **ref** (Optional[int]) –
- **role** (Optional[str]) –
- **result** (Optional[*Result*]) –

4.5 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.**MaxRetriesReached**(*retry_count*, *exceptions*)

Raised if max retries reached and the Overpass server didn't respond with a result.

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.**OverpassError**(*msg=None*)

Base exception to report errors if the response returns a remark tag or element.

Note: If you are not sure which of the subexceptions you should use, use this one and try to parse the message.

For more information have a look at https://github.com/DinoTools/python-overpy/issues/62

Parameters

msg (*str*) – The message from the remark tag or element

msg

The message from the remark tag or element

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.**OverpassRuntimeError**(*msg=None*)

Raised if the server returns a remark-tag(xml) or remark element(json) with a message starting with 'runtime error:'.

exception overpy.exception.**OverpassRuntimeRemark**(*msg=None*)

Raised if the server returns a remark-tag(xml) or remark element(json) with a message starting with 'runtime remark:'.

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.**OverpassUnknownError**(*msg=None*)

Raised if the server returns a remark-tag(xml) or remark element(json) and we are unable to find any reason.

exception overpy.exception.**OverpassUnknownHTTPStatusCode**(*code*)

Raised if the returned HTTP status code isn't handled by OverPy.

Parameters

code (*Integer*) – The HTTP status code

4.6 Helper

overpy.helper.**get_intersection**(*street1*, *street2*, *areacode*, *api=None*)

Retrieve intersection of two streets in a given bounding area

Parameters

- **street1** (str) – Name of first street of intersection
- **street2** (str) – Name of second street of intersection
- **areacode** (str) – The OSM id of the bounding area
- **api** (Optional[*Overpass*]) – API object to fetch missing elements

Return type
List[*Node*]

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

- **street** (str) – Name of street
- **areacode** (str) – The OSM id of the bounding area
- **api** (Optional[*Overpass*]) – API object to fetch missing elements

Return type
Result

Returns
Parsed result

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

CHAPTER

FIVE

CONTRIBUTING

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

5.1 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.

5.2 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:

5.2.1 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.

5.2.2 Documentation

When introducing new functionality, please remember to write documentation.

5.2.3 Tests

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

5.3 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

SIX

CHANGELOG

6.1 0.x (master)

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

6.2 0.6 (2021-04-20)

- Drop Python 2.7 support
- Replace Travis CI with GitHub Actions
- Change ThreadedHTTPServer to HTTPServer for tests
- Add CodeQL CI tests
- Add pre-commit
- Improve build steps

6.3 0.5 (2021-04-14)

- Add support to retry a failed request
- Improve test handlers to fix issues with Py2.7 and PyPy
- Improve tests
- Add Support for Python 3.6
- Add function to handle remark tags and elements
- Add support to parse data from ET.Element
- Move attribute modifiers from class to package
- Change to use Exception instead of BaseException

6.4 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

6.5 0.3.1 (2015-04-30)

- Improve example

6.6 0.3.0 (2015-04-30)

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

6.7 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

6.8 0.1.0 (2014-12-14)

Proof of concept

- Initial release.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

P

Q

R

W