## Contents

1 Breaking Changes 3  
  1.1 Introduction ......................................................... 3  
    1.1.1 What is ORKG .................................................. 3  
    1.1.2 Who can use orkg package .................................. 4  
  1.2 Installation and Usage ............................................. 4  
    1.2.1 Installation .................................................... 4  
    1.2.2 Usage ............................................................ 4  
  1.3 Examples ............................................................. 4  
    1.3.1 Main Components ............................................... 5  
    1.3.2 Individual Components ....................................... 5
ORKG python package is a simple API wrapper for the ORKG’s API.

Start using it by typing these few words:

```bash
$ pip install orkg
```
Breaking Changes

Starting with python package 0.11.0, all paginated calls have new pagination parameters.

<table>
<thead>
<tr>
<th>Old param name</th>
<th>New param name</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>size</td>
</tr>
<tr>
<td>sortBy</td>
<td>sort</td>
</tr>
</tbody>
</table>

The rest of the params stay the same.

1.1 Introduction

This package is just a wrapper for the ORKG’s functionality, it provides an easy access to the API.

1.1.1 What is ORKG

We build the next generation digital libraries for semantic scientific knowledge communicated in scholarly literature. We focus on the communicated content rather than the context e.g., people and institutions in which scientific knowledge is communicated, and the content is semantic i.e., machine interpretable.

Scientific knowledge continues to be confined to the document, seemingly inseparable from the medium as hieroglyphs carved in stone. The global scientific knowledge base is little more than a collection of documents. It is written by humans for humans, and we have done so for a long time. This makes perfect sense, after all it is people that make up the audience, and researchers in particular.

Yet, with the monumental progress in information technologies over the more recent decades, one may wonder why it is that the scientific knowledge communicated in scholarly literature remains largely inaccessible to machines. Surely it would be useful if some of that knowledge is more available to automated processing.

The Open Research Knowledge Graph project is working on answers and solutions. The recently initiated TIB coordinated project is open to the community and actively engages research infrastructures and research communities in the development of technologies and use cases for open graphs about research knowledge.
For more information about the ORKG, please check the project page, and the application page.

1.1.2 Who can use orkg package

Well, the short answer is anybody. The package is designed to be minimalistic, simple knowledge of python and JSON suffices for you to know and understand how the package works.

You can use the ORKG package to add/edit/list data from any instance of the open research knowledge graph. It can be easily integrated into data science workflows, it can be used to fetch data for analysis and visualizations.

The sky is your limit :)

1.2 Installation and Usage

The wrapper is pretty lightweight and doesn’t need any requirements.

1.2.1 Installation

The package can be installed using pip

```
$ pip install orkg
```

Or the source can be cloned from:

```
$ git clone https://gitlab.com/TIBHannover/orkg/orkg-pypi.git
```

1.2.2 Usage

In order to use the package in your python code, you just need to import it and instantiate an instance of the base class to use it.

```
from orkg import ORKG # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password')) # ←create the connector to the ORKG
```

The package can be used to connect to any instance of the ORKG local or remote. Optionally you can provide the credentials to authenticate your requests.

1.3 Examples

Two main classes should be known when using the ORKG python package.

- ORKG class (main class to connect to an ORKG instance).
- OrkgResponse (output encapsulation for the ORKG API).
1.3.1 Main Components

To instantiate an ORKG connector, you just add two lines:

```python
from orkg import ORKG  # import base class from package
connector = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))  
    # create the connector to the ORKG
```

The connector has many properties for facilitate access of specific features of the API:

```python
connector.resources  # entry point to manipulate ORKG resources
connector.predicates  # entry point to manipulate ORKG predicates
connector.classes  # entry point to manipulate ORKG classes
connector.literals  # entry point to manipulate ORKG literals
connector.stats  # entry point to get ORKG statistics
connector.statements  # entry point to manipulate ORKG statements
connector.papers  # entry point to manipulate ORKG papers
connector.comparisons  # entry point to manipulate ORKG comparisons
```

For the other main component which represent the output of all requests in the package:

```python
connector.resources.get()  # this call will fetch a collection of resources from the ORKG

>>> (Success) [{'id': 'R0', 'label': "Gruber's design of ontologies", 'created_at': '2019-01-06T15:04:07.692Z', 'classes': [], 'shared': 1, 'created_by': '00000000-0000-0000-0000-000000000000', '_class': 'resource'}, {'id': 'R172', 'label': 'Oceanography', 'created_at': '2019-01-06T15:04:07.692Z', 'classes': ['ResearchField'], 'shared': 1, 'created_by': '00000000-0000-0000-0000-000000000000', '_class': 'resource'}, ...]
```

Internally the OrkgResponse class contains information about the request and response from the API which can be accessed.

```python
response = connector.resources.get()  # get OrkgResponse object
response.succeeded  # returns bool if the call is a success or not
>>> True
response.status_code  # get the HTTP status code of the request
>>> 200
response.url  # get url of the request
>>> https://orkg.org/api/resources/
response.content  # get response content as dict (JSON object) or list of dicts (JSON array)

>>> [{'id': 'R0', 'label': "Gruber's design of ontologies", 'created_at': '2019-01-06T15:04:07.692Z', 'classes': [], 'shared': 1, 'created_by': '00000000-0000-0000-0000-000000000000', '_class': 'resource'}, {'id': 'R172', 'label': 'Oceanography', 'created_at': '2019-01-06T15:04:07.692Z', 'classes': ['ResearchField'], 'shared': 1, 'created_by': '00000000-0000-0000-0000-000000000000', '_class': 'resource'}, ...]
```

1.3.2 Individual Components

For the individual components, they follow the same calls that are supported by the ORKG API.
ORKG, Release 0.11.3

For detailed list of all components

**ORKG Resources**

Entities in the ORKG can be of type *Resource*, in order to be able to access this information or manipulate it (i.e., add, edit, delete) a *resources* component is added to the ORKG class to encapsulate the actions.

Having defined our entry point to the ORKG instance

```python
from orkg import ORKG  # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))  # create the connector to the ORKG
```

We can access the resources manager directly to do the following:

**Getting resources by ID**

You can get certain resources given that you know their ID value

```python
### Fetch resource by id
orkg.resources.by_id(id='R0')
```

```python
>>> (Success)
{
   "id":"R0",
   "label":"Gruber’s design of ontologies",
   "created_at":"2019-01-06T15:04:07.692Z",
   "classes": [],
   "shared":1,
   "created_by":"00000000-0000-0000-0000-000000000000",
   "_class":"resource"
}
```

**Getting resources by lookup**

You can lookup a resource by the label of the resource. A variety of parameter can be passed to specify an exact match, number of results to retrieve, and what order you need them in.

```python
### Fetch a list of resources
orkg.resources.get(q='ORKG', exact=False, size=30, sort='label', desc=True)
```

```python
>>> (Success)
[
   {
     "id":"R125",
     "label":"ORKG system",
     "created_at":"2019-01-06T15:04:07.692Z",
     "classes": [ ]
   }
]
```

(continues on next page)
Adding a new resource

The ORKG package can be used to create new resources in the ORKG instance you are connected to.

*Note: if you have your credentials entered in the ORKG instance creation all newly added resources will be credited to your user.*

```python
### Adds a resource to the graph
# id: the id of the resource (optional)
# label: the label of the resource
# classes: the list of classes that a resource belong to (optional)
orkg.resources.add(id='Custom_ID', label='new resource', classes=['Paper'])
```

```json
{
  "id" : "Custom_ID",
  "label" : "new resource",
  "created_at" : "2019-01-06T15:04:07.692Z",
  "classes" : [
    "Paper"
  ],
  "shared" : 1,
  "created_by" : "00000000-0000-0000-0000-000000000000",
  "_class" : "resource"
}
```

Updating an existing resource

You can also update an existing resource in the ORKG other than creating a new one. (*Use carefully*)

```python
### Updates a resource
# id (mandatory): the id of the resource to change
# label: the new label of the resource
# classes: the list of new classes (optional)
orkg.resources.update(id='R1', label='ORKG is so cool', classes=[])}
```

```json
{
  "id" : "R1",
  "label" : "ORKG is so cool",
  "created_at" : "2019-01-06T15:04:07.692Z",
  "classes" : [
  ],
  "shared" : 1,
  "created_by" : "00000000-0000-0000-0000-000000000000",
  "_class" : "resource"
}
```

1.3. Examples
Check if a resource exist

For your code to run smoothly you can check for the existence of resource before you update them for example. You can make sure that you code doesn’t run into unexpected results.

```python
### Checks if resource exists
# id: the id of the resource
# returns a bool
orkg.resources.exists(id='R1')
```

Create a resource of tabular data

This function is provided in regard to support publishing statistical data in ORKG, and as a first step, we support two-dimensional tables like CSV files. And we are using the RDF Data Cube Vocabulary to model the dataset inside the graph.

The function has three parameters:
1- file: csv file containing the table
2- label: the label of the output resource
3- dimensions: a list of column(s) name(s) used to identify the observations (ex. time, region), what the observation applies to.

*Note: The columns that are not in the dimensions list are considered as measures (the phenomenon being observed) and defining attribute(s) is not supported yet!*

Behind the scenes there is a three-step process:
1- Parse the CSV file using the pandas library
2- Create Data Structure Definition
3- Create the observation resource for each row (the actual data in the cells of a table).

The final output is an ORKG Resource ID that can be used in defining contribution data of papers. And it has a tabular representation in the front-end.

Example:

Let’s imagine this simple CSV file ‘example.csv’ where the first column is the dimension:

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Event start time</th>
<th>Event end time</th>
<th>Duration</th>
<th>Sunrise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>09:05</td>
<td>14:25</td>
<td>04:41</td>
<td>04:34</td>
</tr>
<tr>
<td>Max</td>
<td>12:17</td>
<td>18:41</td>
<td>06:32</td>
<td>07:50</td>
</tr>
</tbody>
</table>

To create a resource for this table in ORKG we call the function:

```python
### Create a resource of a tabular data using RDF Dacucube vocabulary starting from a CSV file
# file: CSV file containing the table
# label: label of the resource
# dimensions: a list of column(s) name(s) that represent the dimensions.
datasetID = save_dataset(file="example.csv", label="Summary", dimensions=["Statistics"])
```

(continues on next page)
ORKG Literals

Entities in the ORKG can be of type *Literal*, this means they represent the ground values (strings, numbers, dates), in order to be able to access this information or manipulate it (i.e., add, edit, delete) a *literals* component is added to the ORKG class to encapsulate the actions.

Having defined our entry point to the ORKG instance

```python
from orkg import ORKG  # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))  # create the connector to the ORKG
```

We can access the literals manager directly to do the following:

**Getting literal by ID**

You can get certain literals given that you know their ID value

```python
### Fetch literal by id
orkg.literals.by_id(id='L1')
```

```json
>>> (Success)
{
    "id":"L1",
    "label":"ORKG is the best project ever",
    "created_at":"2019-01-06T15:04:07.692Z",
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"literal"
}
```

**Getting literals by lookup**

You can lookup literals by the value of the literal.

```python
### Get all literals (Not paginated)
# all the parameters are optional
orkg.literals.get_all(q='ORKG', exact=False)
```

```json
>>> (Success)
[{
    "id":"L1",
    "label":"ORKG is the best project ever",
    "created_at":"2019-01-06T15:04:07.692Z",
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"literal"
}, ...
]```
Adding a new literal

The ORKG package can be used to create new literals in the ORKG instance you are connected to.

*Note: if you have you credentials entered in the ORKG instance creation all newly added literals will be credited to your user.*

```python
### Add a literal
# ID is optional
orkg.literals.add(label='Science rocks!')
```  
```json
>>> (Success)
{
    "id": "L1234",
    "label": "Science rocks!",
    "created_at": "2019-01-06T15:04:07.692Z",
    "created_by": "00000000-0000-0000-0000-000000000000",
    "_class": "literal"
}
```

Updating an existing literal

You can also update an existing literal in the ORKG other than creating a new one. (*Use carefully*)

```python
### Update a literal
# not available on Labs yet
orkg.literals.update(id='L1', label='Coco')
```  
```json
>>> (Success)
{
    "id": "L1",
    "label": "Coco",
    "created_at": "2019-01-06T15:04:07.692Z",
    "created_by": "00000000-0000-0000-0000-000000000000",
    "_class": "literal"
}
```

Check if a literal exists

For your code to run smoothly you can check for the existence of literals before you update them for example. You can make sure that you code doesn’t run into unexpected results.

```python
### Checks if literal exists
# id: the id of the literal
# returns a bool
orkg.literals.exists(id='L1')
```  
```python
>>> True
```

ORKG Predicates

Entities in the ORKG can be of type *Predicate*, in order to be able to access this information or manipulate it (i.e., add, edit, delete) a *predicates* component is added to the ORKG class to encapsulate the actions.

Having defined our entry point to the ORKG instance
from orkg import ORKG  # import base class from package

orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))  # create the connector to the ORKG

We can access the predicates manager directly to do the following:

### Getting predicates by ID

You can get certain predicate given that you know their ID value

```python
>>> (Success)
{
  "id": "P1",
  "label": "yields",
  "created_at": "2019-01-06T15:04:07.692Z",
  "created_by": "00000000-0000-0000-0000-000000000000",
  "_class": "predicate"
}
```

### Getting predicates by lookup

You can lookup a predicate by the label of it. A variety of parameter can be passed to specify an exact match, number of results to retrieve, and what order you need them in.

```python
>>> (Success)
[
  {
    "id": "P1",
    "label": "yields",
    "created_at": "2019-01-06T15:04:07.692Z",
    "created_by": "00000000-0000-0000-0000-000000000000",
    "_class": "predicate"
  },
  ...
]
```

### Adding a new predicate

The ORKG package can be used to create new predicates in the ORKG instance you are connected to.

*Note: if you have you credentials entered in the ORKG instance creation all newly added predicates will be credited to your user.*
### Adds a predicate to the graph

`orkg.predicates.add(id='Custom_ID', label='new predicate')`

```python
>>> (Success)
{
    "id":"Custom_ID",
    "label":"new predicate",
    "created_at":"2019-01-06T15:04:07.692Z",
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"predicate"
}
```

### Updating an existing predicate

You can also update an existing predicates in the ORKG other than creating a new one. (*Use carefully*)

```python
>>> orkg.predicates.update(id='Custom_ID', label='the newest predicate!')
>>> (Success)
{
    "id":"Custom_ID",
    "label":"the newest predicate!",
    "created_at":"2019-01-06T15:04:07.692Z",
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"predicate"
}
```

### Check if a predicate exist

For your code to run smoothly you can check for the existence of predicates before you update them for example. You can make sure that you code doesn’t run into unexpected results.

```python
>>> orkg.predicates.exists(id='P112313')
>>> False
```

### ORKG Classes

Resources in the ORKG are typed to one or more of the ORKG’s classes, in order to be able to access this information or manipulate it (i.e., add, edit, delete) a classes component is added to the ORKG class to encapsulate the actions.

Having defined our entry point to the ORKG instance

```python
from orkg import ORKG  # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))  # create the connector to the ORKG
```

Chapter 1. Breaking Changes
We can access the classes manager directly to do the following:

**Getting class by ID**

You can get certain class given that you know their ID value

```python
### Fetch class by id
# id: the class id
orkg.classes.by_id(id='C5')
>>> (Success)
{
  "id":"C5",
  "label":"MeasureProperty",
  "uri":None,
  "created_at":"2020-05-11T14:52:34.316361+02:00",
  "created_by":"00000000-0000-0000-0000-000000000000",
  "_class":"class"
}
```

**Getting classes by lookup**

You can lookup a class by it’s label. You can set a parameter to specify an exact match or a loose one.

```python
### Fetch a list of predicates
# All parameters are optional
# q: to search for a label containing
# exact: to condition the search to as is
orkg.classes.get_all(q='Property', exact=False)
>>> (Success)
[{
  "id":"C5",
  "label":"MeasureProperty",
  "uri":None,
  "created_at":"2020-05-11T14:52:34.316361+02:00",
  "created_by":"00000000-0000-0000-0000-000000000000",
  "_class":"class"
},
...
]
```

**Adding a new class**

The ORKG package can be used to create new classes in the ORKG instance you are connected to.

*Note: if you have you credentials entered in the ORKG instance creation all newly added predicates will be credited to your user.*

```python
### Add a class
# id: the id of the class (optional)
# label: the label of the class
# uri: the external uri that is equivalent to this class from an ontology or a
→taxonomy (optional)
```

(continues on next page)
orkg.classes.add(id='Custom_ID', label='new class', uri='http://external-vocab.com')

>>> (Success)
{
    "id":"Custom_ID",
    "label":"new class",
    "uri":"http://external-vocab.com",
    "created_at":"2020-05-11T14:52:34.316361+02:00",
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"class"
}

Getting all resources of a certain class

You can use the class manager of the ORKG python package to get resources of a certain class.

```python
>>> orkg.classes.get_resource_by_class(class_id='Paper', size=30, sort='label', desc=True, q='Paper', exact=False)
{
    "id":"R1223",
    "label":"Paper 1",
    "created_at":"2019-01-06T15:04:07.692Z",
    "classes":[
        "Paper"
    ],
    "shared":1,
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"resource"
}
...
```

Updating an existing class

You can also update an existing class in the ORKG other than creating a new one. (*Use carefully*)

```python
>>> orkg.classes.update(id='Custom_ID', label='Coco')
{
    "id":"Custom_ID",
    "label":"Coco",
    "uri":"http://external-vocab.com",
    "created_at":null,
    "created_by":null,
    "updated_at":null,
    "updated_by":null,
    "_class":"class"
}
```
Check if a class exists

For your code to run smoothly you can check for the existence of classes before you update them for example. You can make sure that your code doesn’t run into unexpected results.

```python
# Checks if class exists
# id: the id of the class
# returns a bool
orkg.classes.exists(id='C1')
```

### ORKG Statements

Entities in the ORKG are connected to each other via predicates, the is struct of tuples <subject, predicate, object> is referred to as a statement, in order to be able to access this information or manipulate it (i.e., add, edit, delete) a statements component is added to the ORKG class to encapsulate the actions.

Having defined our entry point to the ORKG instance

```python
from orkg import ORKG # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password')) # create the connector to the ORKG
```

We can access the statements manager directly to do the following:

### Getting statement by ID

You can get certain statements given that you know their ID value

```python
# Fetch statement by id
# id: the statement id
orkg.statements.by_id(id='S5')
```

(continues on next page)
Getting statements

You can get a list of statements. A variety of parameter can be passed to specify what order you need them in and how many results.

```python
### Get all statements
#
# all the parameters are optional
# size: to specify the number of items in the page
# sort: to specify the key to sort on
# desc: to set the direction of sorting
orkg.statements.get(size=30, sort='id', desc=True)
```

>>> (Success)

```json
[
 {
   "id":"S5",
   "subject":{
      "id":"R0",
      "label":"Gruber's design of ontologies",
      "created_at":"2019-01-06T15:04:07.692Z",
      "classes":[
      ],
      "shared":1,
      "created_by":"00000000-0000-0000-0000-000000000000",
      "_class":"resource"
   },
   "predicate":{
      "id":"P0",
      "label":"addresses",
      "created_at":"2020-05-11T15:06:23.588208+02:00",
      "created_by":"00000000-0000-0000-0000-000000000000",
      "_class":"predicate"
   },
   "object":{
      "id":"R8",
      "label":"Design of ontologies",
      "created_at":"2019-01-06T15:04:07.692Z",
      "created_by":"00000000-0000-0000-0000-000000000000"
   }
 }]
```
Getting statements by subject

You can a list of statements that have a specific subject

```bash
### Get all statements by subject
# subject_id is the subject to filter on
# other parameters are optional
orkg.statements.get_by_subject(subject_id='R0', size=30, sort='id', desc=True)
>>> (Success)
[
    {
        "id":"S5",
        "subject":{
            "id":"R0",
            "label":"Gruber's design of ontologies",
            "created_at":"2019-01-06T15:04:07.692Z",
            "classes":[
            ],
            "shared":1,
            "created_by":"00000000-0000-0000-0000-000000000000",
            "_class":"resource"
        },
        "predicate":{
            "id":"P0",
            "label":"addresses",
            "created_at":"2020-05-11T15:06:23.588208+02:00",
            "created_by":"00000000-0000-0000-0000-000000000000",
            "_class":"predicate"
        },
        "object":{
            "id":"R8",
            "label":"Design of ontologies",
            "created_at":"2019-01-06T15:04:07.692Z",
            "classes":[
            ],
            "shared":1,
            "created_by":"00000000-0000-0000-0000-000000000000",
            "_class":"resource"
        },
        "created_at":"2019-01-06T15:04:07.692Z",
        "created_by":"00000000-0000-0000-0000-000000000000"
    }
]```
Getting statements by predicate

You can a list of statements that have a specific predicate

```python
### Get all statements by predicate
# predicate_id is the predicate to filter on
# other parameters are optional
orkg.statements.get_by_predicate(predicate_id='P0', size=20, sort='id', desc=True)

>>> (Success)
[
    {
        "id":"S5",
        "subject":{
            "id":"R0",
            "label":"Gruber's design of ontologies",
            "created_at":"2019-01-06T15:04:07.692Z",
            "classes":[
            ],
            "shared":1,
            "created_by":"00000000-0000-0000-0000-000000000000",
            "_class":"resource"
        },
        "predicate":{
            "id":"P0",
            "label":"addresses",
            "created_at":"2020-05-11T15:06:23.588208+02:00",
            "created_by":"00000000-0000-0000-0000-000000000000",
            "_class":"predicate"
        },
        "object":{
            "id":"R8",
            "label":"Design of ontologies",
            "created_at":"2019-01-06T15:04:07.692Z",
            "classes":[
            ],
            "shared":1,
            "created_by":"00000000-0000-0000-0000-000000000000",
            "_class":"resource"
        },
        "created_at":"2019-01-06T15:04:07.692Z",
        "created_by":"00000000-0000-0000-0000-000000000000"
    },
    ...
]
```

Getting statements by object

You can a list of statements that have a specific object
### Get all statements by object

Get all statements by a specific object.

```python
orkg.statements.get_by_object(object_id='R8', size=5, sort='id', desc=False)
```

```python
>>> (Success)
[
  {
    "id":"S5",
    "subject":{
      "id":"R0",
      "label":"Gruber's design of ontologies",
      "created_at":"2019-01-06T15:04:07.692Z",
      "classes":[]
    },
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"resource"
  },
  "predicate":{
    "id":"P0",
    "label":"addresses",
    "created_at":"2020-05-11T15:06:23.588208+02:00",
    "created_by":"00000000-0000-0000-0000-000000000000",
    "_class":"predicate"
  },
  "object":{
    "id":"R8",
    "label":"Design of ontologies",
    "created_at":"2019-01-06T15:04:07.692Z",
    "classes":[]
  },
  "created_at":"2019-01-06T15:04:07.692Z",
  "created_by":"00000000-0000-0000-0000-000000000000"
},
...
]
```

Adding a new statement

The ORKG package can be used to create new statements in the ORKG instance you are connected to.

**Note:** if you have you credentials entered in the ORKG instance creation all newly added statements will be credited to your user.

### Add a statement

Add a statement with specific subject, predicate, and object.

```python
orkg.statements.add(subject_id='R1', predicate_id='P2', object_id='L3')
```

```python
>>> (Success)
{
  "id":"S32131",
  ...
}
```
Updating an existing statement

You can also update an existing statements in the ORKG other than creating a new one. (*Use carefully*)

```python
>>> orkg.statements.update(id='S32131', subject_id='R1', predicate_id='P3', object_id='L3')
(Success)
```

>>> (Success)
```json
"id":"S32131",
"subject":{
    "id":"R1",
    "label":"Some resource",
    "created_at":"2019-01-06T15:04:07.692Z",
    "classes":[
    ],
    "shared":1,
    "created_by":"00000000-0000-0000-0000-000000000000",
    ",
    "_class":"resource"
},
"predicate":{
    "id":"P3",
    "label":"some other predicate!",
    "created_at":"2020-05-11T15:06:23.588208+02:00",
    "created_by":"00000000-0000-0000-0000-000000000000",
    ",
    "_class":"predicate"
},
"object":{
    "id":"L3",
    "label":"some literal",
    "created_at":"2019-01-06T15:04:07.692Z",
    "created_by":"00000000-0000-0000-0000-000000000000",
    ",
    "_class":"literal"
}
```
Check if a statement exist

For your code to run smoothly you can check for the existence of statements before you update them for example. You can make sure that you code doesn’t run into unexpected results.

```python
## Checks if statement exists
# id: the id of the statement
# returns a bool
orkg.statements.exists(id='S1')
```

### ORKG Papers

One of the main power points of the ORKG is the ability to represent scientific papers in a structured manner. The package also provide an endpoint to add new papers directly using the concepts defined on other components.

A paper technically is a collection of resources, literals, and statements connecting them in a small sub-graph.

Having defined our entry point to the ORKG instance

```python
from orkg import ORKG # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))
```

We can access the papers manager directly to do the following:

#### Adding new papers

The ORKG package can be used to create new papers in the ORKG instance you are connected to.

Note: if you have you credentials entered in the ORKG instance creation all newly added papers will be credited to your user.

```python
# First define the paper content as dict (JSON object) (This is a sample of input)
paper = {
    "predicates": [
    ],
```
"paper": {
  "title": "Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge",
  "doi": "10.1145/3360901.3364435",
  "authors": [
    { "label": "Mohamad Yaser Jaradeh" },
    { "label": "Allard Oelen" },
    { "label": "Kheir Eddine Farfar" },
    { "label": "Manuel Prinz" },
    { "label": "Jennifer D'Souza" },
    { "label": "Gábor Kismihók" },
    { "label": "Markus Stocker" },
    { "label": "Sören Auer" }
  ],
  "publicationMonth": "",
  "publicationYear": 2019,
  "publishedIn": "Proceedings of the 10th International Conference on Knowledge Capture - K-CAP '19",
  "researchField": "R11",
  "contributions": [
    { "name": "Contribution 1",
      "values": {
        "P32": [
          { "@temp": "_5254e420-ae9a-13ef-1c18-e716b9ea5c2b",
            "label": "Similarity measures",
            "class": "Problem",
            "values": {} }
        ],
        "P3": [
          { "text": "Find similar research contributions inside the ORKG and suggest them to the user" }
        ],
        "P1": [
          ...
        ]
      }
    }
  ]
}
(continued from previous page)

```python
orkg.papers.add(params=paper)
>>> (Success)
{
  "id":"R8186",
  "label":"Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge",
  "created_at":"2019-12-19T15:07:02.136+01:00",
  "classes":[
    "Paper"
  ],
  "shared":0,
  "created_by":"00000000-0000-0000-0000-000000000000",
  "_class":"resource"
}

# you can append the contribution into the paper if it exists in the system by specifying the 'merge_if_exists' property
orkg.papers.add(params=paper, merge_if_exists=True)
>>> (Success)
{
  "id":"R8186",
  "label":"Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge",
  "created_at":"2019-12-19T15:07:02.136+01:00",
  "classes":[
    "Paper"
  ],
  "shared":0,
  "created_by":"00000000-0000-0000-0000-000000000000",
  "_class":"resource"
}
```

Adding new papers via CSV

An easier way to import papers from a csv file containing the details is detailed in this Wiki page.

1.3. Examples
ORKG Objects

**Warning:** Should be used by power-users only. This endpoint could have unaccounted consequences.

The package provides an endpoint to add new objects directly using the concepts defined on other components. An object is a collection of resources, literals, and statements connecting them in a small sub-graph.

Having defined our entry point to the ORKG instance

```python
from orkg import ORKG # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password')) # create the connector to the ORKG
```

We can access the objects manager directly to do the following:

### Adding new object

The ORKG package can be used to create new objects in the ORKG instance you are connected to.

*Note: if you have you credentials entered in the ORKG instance creation all newly added papers will be credited to your user.*

```python
# First define the object content as dict (JSON object) (This is a sample of input)
obj = {
    "predicates": [],
    "resource": {
        "name": "I rock maybe!",
        "classes": [
            "C2000"
        ],
        "values": {
            "P32": [
                {
                    "@id": "R2"
                }
            ],
            "P55": [
                {
                    "label": "ORKG is so cool!",
                    "classes": [
                        "C3000"
                    ]
                }
            ]
        }
    }
}

# Then add the content as an object to the ORKG instance
orkg.objects.add(params=obj)
```

```python
>>> (Success)
```

(continues on next page)
ORKG Contributions client

The contribution client is a special client of the ORKG that handles contributions and comparisons. For that reason it needs to be connected to the simcomp service of the open research knowledge graph.

Doing so is as simple as instatiation the ORKG client but with providing the simcomp_host param

```python
from orkg import ORKG  # import base class from package
orkg = ORKG(host="<host-address-is-here>", simcomp_host="<simcomp-host-address>", creds=('email-address', 'password'))  # create the connector to the ORKG
```

We can access the contributions client directly to do the following:

Get similar contributions

With the following line of code you can easily create a dummy response.

```python
>>> orkg.contributions.similar(cont_id='R8197')
```

```json
{
   "contributionId":"R6142",
   "contributionLabel":"Contribution 1",
   "paperId":"R6141",
   "similarityPercentage":0.13443964521673085
},
{
   "contributionId":"R8199",
   "contributionLabel":"ORKG System",
   "paperId":"R8186",
   "similarityPercentage":0.1338162327194794
}
```

Comparing several contributions

You can perform a comparison of several contributions easily as well.

1.3. Examples
orkg.contributions.compare(contributions=['R6151','R6173','R6162','R6179','R6157',
' R6146'])

>>> (Success)
{
    "contributions": [
        {
            "contributionLabel": "Contribution 1",
            "id": "R6151",
            "paperId": "R6150",
            "title": "Author name disambiguation using a graph model with node splitting and merging based on bibliographic information",
            "year": "2014"
        },
        ...
    ],
    "data": {
        "METHOD": [
            {
                "label": "Graph Node Splitting & Merging",
                "path": [
                    "R6151",
                    "METHOD"
                ],
                "pathLabels": [
                    "Contribution 1",
                    "Method"
                ],
                "resourceId": "L4214",
                "type": "literal"
            },
            {
                "label": "Pairwise Factor Graph",
                "path": [
                    "R6173",
                    "METHOD"
                ],
                "pathLabels": [
                    "Contribution 1",
                    "Method"
                ],
                "resourceId": "R6163",
                "type": "resource"
            },
            {
                "label": "interaction of user to enhance the performance",
                "path": [
                    "R6173",
                    "METHOD"
                ],
                "pathLabels": [
                    "Contribution 1",
                    "Method"
                ],
                "resourceId": "L4239",
            }
        ]
    }
}

(continues on next page)
Comparisons as pandas.DataFrame

To make it easy to process the content of the comparison and not worry about the JSON format that is returned from the backend.

You can easily deal with the data as pandas.DataFrame object that you can use in down-stream applications.

```python
>>> orkg.contributions.compare(contributions=['R6151', 'R6173', 'R6162', 'R6179', 'R6157', 'R6146'], response_hash='972c8cef92829c1fa5c3b43caf5709c7')
>>> (Success)
```

## Same response as before

### One of the parameters is required

#### Using the comparison ID

```python
df = orkg.contributions.compare_data_frame(comparison_id='R6751')
type(df)
```

```text
<
pandas.core.frame.DataFrame
```

#### Using the list of contributions

```python
orkg.contributions.compare_data_frame(contributions=['R6151', 'R6173', 'R6162', 'R6179', 'R6157', 'R6146'])
type(df)
```

```text
<
pandas.core.frame.DataFrame
```

(continues on next page)
ORKG JSON client

**Warning:** Should be used by power-users only. This endpoint could have unaccounted consequences.

The SimComp service has an endpoint to store json objects to enable other services performing persistence features. You need to have the `simcomp_host` defined to be able to use this client.

```python
from orkg import ORKG  # import base class from package
orkg = ORKG(host="<host-address-is-here>", simcomp_host="<simcomp-host-address>", creds=('email-address', 'password'))  # create the connector to the ORKG
```

We can access the json client directly to do the following:

**Save JSON objects**

With the following line of code you can easily create a dummy response.

```python
### all the parameters are required
orkg.json.save_json(resource_id='R22225', json_object={'key': 'value'})
```

```python
>>> (Success)
{
  "success":true
}
```

**Load JSON objects**

You can perform a comparison of several contributions easily as well.

```python
orkg.json.get_json(resource_id='R22225')
```

```python
>>> (Success)
{
  "key": "value"
}
```

**ORKG Dummy client**

To create dummy `OrkgResponse` objects, this dummy client provides the following functionality

```python
from orkg import ORKG  # import base class from package
orkg = ORKG(host="<host-address-is-here>", creds=('email-address', 'password'))  # create the connector to the ORKG
```

We can access the dummy data manager directly to do the following:
Creating a 200 response

With the following line of code you can easily create a dummy response.

```python
>>> orkg.dummy.create_200_response(content={'orkg': 'super easy'})
(Success)
{
    "orkg": "super easy"
}
```

Creating a 404 response

You can lookup literals by the value of the literal.

```python
>>> orkg.dummy.create_404_response(content=[{'msg': 'bad content'}])
(Fail)
[
    {
        "msg": "bad content"
    },
    ...
]
```

More generic responses

You can create any XXX HTTP status code response to match your testing or mocking needs via the following function:

```python
>>> orkg.dummy.create_xxx_response(code='500', content={'error': 'server died'})
(Fail)
{
    "error": "server died"
}
```