diggrtoolbox Documentation

F. Rämisch and P. Mühleder

May 22, 2023

Contents:

diggrtoolbox is a collection of various loosely coupled or completely independent tools, which were developed during the first phase of the diggr (databased infrastructure for global game culture research) project at the university library in Leipzig.

The tools are mostly small helpers meant to ease the handling of data and data structures we encountered during this research project.

Note: The main development paradigm for this library was and is: Providing tools, which have few to no additional/external dependencies, especially no requirement for any services to be run in the network, e.g. elasticsearch, CouchDB, etc. It is a toolbox made for Digital Humanities Researchers who do not have access to a huge technical infrastructure.

CHAPTER 1

Getting started

1.1 diggrtoolbox

This collection of tools was developed in the Databased infrastructure for global game culture reasearch (diggr) group at the University Library in Leipzig. Being a collection means, that these helpers are organised into individual packages. Each package is built for one purpose, but the functionality and purpose across functionality may be differ.

For the full documentation have a look at https://diggrtoolbox.readthedocs.io

1.1.1 Requirements

This Software was tested with Python 3.5 and 3.6. There are no further requirements. *diggrtoolboxes* uses only packages and modules which are shipped with Python. Only exception: If you plan development on *diggrtoolbox* you need to have *pytest* to run the tests.

1.1.2 Components

- *deepget*: A small helper easing access to data in deeply nested dicts/list, by separating the definition of the route and actual call.
- *ZipSingleAccess*: Allows access to a JSON document in a ZIP-File.
- *ZipMultiAccess*: Allows access to a JSON document in a ZIP-File, where some parts of the original JSON document are separated into separate json documents. This eases the handling of large files, which otherwise would clog the RAM.
- *TreeExplore*: Class to help exploring deeply nested dicts/lists/both. It provides various helpful display and search functions. It can help exploring raw dumps aquired from APIs on the internet. The search function returns a route-object which can be fed to deepget, in order to retrieve specific datasets.
- *treehash*: Allows comparison of complex data structures by hashing it. It allows to compare deeply nested dicts/lists/both without having to compare its individual components.

1.1.3 Authors

- Florian Rämisch <raemisch@ub.uni-leipzig.de>
- Peter Mühleder <muehleder@ub.saw-leipzig.de>

1.1.4 License

- MIT License.

1.1.5 Copyright

1.2 Installation

It is recommended to use *diggrtoolbox* in a virtualenvironment such as virtualenv. Please refer to the documentation of virtualenv and/or virtualenvwrapper or pipenv to see how to set it up.

The latest version of *diggrtoolbox* can be obtained from github.

1.2.1 Install the latest version

You can install the latest version via pip:

```
pip install git+https://github.com/diggr/diggrtoolbox
```

1.2.2 Development

If you plan to develop *diggrtoolbox* it is recommended to clone the github repository:

```
git clone git@github.com/diggr/diggrtoolbox
```

Installation is performed using pip, but in editable mode, i.e. such that changes in the source take effect immediately:

```
pip install -e ./diggrtoolbox
```

1.3 Examples

To demonstrate possible applications of the tools of the toolbox, this page will contain example use cases.

1.3.1 UnifiedAPI / DiggrAPI

This is the latest addition to the toolbox. It allows the user to have an easier access to the unifiedAPI without having to memorize addresses. You can set filters, select datasets, etc.

The following will create an instance, and select the dataset mobygames.

```
>>> from diggrtoolbox.unified_api import DiggrAPI
>>> d = DiggrAPI("http://localhost:6660).dataset("mobygames")
```

If you now get() this, you will get a list of all ids.

```
>>> ids = d.get()
```

Let's suppose you are interested in links. Apply a filter, and then iterate over all ids, and run your process

```
>>> d.filter("links")
>>> for id_ in ids:
>>>     data = d.item(id_).get()
>>>     # further processing
```

To clean up the code a bit, you can get the result immediately after setting an item id (or slug), by initializing DiggrAPI with *get_on_item=True*. If the "magic" (i.e. filtering the content of the request instead of returning the raw response) does not fit your needs, you can also set *raw=True*.

```
>>> d = DiggrAPI("http://localhost:6660", get_on_item=True, raw=True)
>>> d.dataset("mobygames").filter("links")
>>> raw_data = d.item("id_")
```

1.3.2 ZipSingleAccess

Imagine you have a lot of data stored in one JSON-file. Often these files can be compressed to take a lot less space on your harddrive. When you want to work with the content of these files, of course you don't want to upack them first:

```
>>> import diggrtoolbox as dt
>>> z = ZipSingleAccess("data/compressed_file.zip")
>>> j = z.json()
>>> isinstance(j, dict)
True
>>> print(j.keys())
dict_keys(['id', 'data', 'raw'])
```

1.3.3 ZipMultiAccess

Sometimes the data, you want so load from a file, which is bigger than the RAM you have. This is a problem, as it makes it impossible to work with files of this size without some tricks.

In the natural sciences this problem is tackled by using HDF5, a special file format, allowing to partially load the file, and only serve the parts needed for the next computation step. Unfortunately, this file is not quite made to store tree like structures like nested dicts/lists.

With ZipMultiAccess we make the first step into this direction. You save subtrees of your data in a subfolder, and then load it from the ZIP when you need it:

```
>>> import diggrtoolbox as dt
>>> z = ZipMultiAccess("data/compressed_files.zip")
>>> j = z.json()
>>> isinstance(j, list)
True
>>> len(j)
38386
```

(continues on next page)

(continued from previous page)

```
>>> isinstance(j[0], dict)
True
>>> print(j[0].keys())
dict_keys(['id', 'data', 'raw', 'matches'])
>>> print(j[0]['matches'])
{'n_matches': 3}
>>> m1 = z.get(j[0]['id'])
>>> isinstance(m, list)
True
>>> len(m)
3
```

In the above example we have a list of 38386 which we matched with other games from another database. The match data is huge, so putting all data into one file resulted in a big freeze, as the amount of memory required to hold put all information into one Python object was larger, than the amount the machine had available.

All match data was put into separate files, in a subfolder *matches* and then referenced with the id in the filename. The name of the subfolder can be chosen arbitrarily.

There are multiple ways of accessing the additional files:

```
>>> z[j[0]['id']] == z.get(j[0]['id'])
True
```

1.3.4 TreeExplore

The TreeExplore class provides easy access to nested dicts/list or combinations of both:

```
>>> import diggrtoolbox as dt
>>> test_dict = {'id' : 123456789,
>>>              'data' : {'name': 'diggr project',
>>>                        'city': 'Leipzig',
>>>                        'field': 'Video Game Culture'},
>>>              'references':[{'url': 'http://diggr.link',
>>>                             'name': 'diggr website'},
>>>                             {'url': 'http://ub.uni-leipzig.de',
>>>                              'name': 'UBL website'}]}
>>> tree = dt.TreeExplore(test_dict)
>>> results = tree.search("leipzig")
Search-Term: leipzig
Route: references, 1, url,
Embedding: 'http://ub.uni-leipzig.de'
>>> print(results)
[{'embedding': 'http://ub.uni-leipzig.de',
  'route': ['references', 1, 'url'],
  'unique_in_embedding': False,
  'term': 'leipzig'}]
```

1.3.5 treehash

Imagine you have a datastructure, which you use as a reference at some point in your workflow. It is provided as a JSON-file at some point online, e.g. the diggr platform mapping for the MediaartsDB.

This file is updated frequently. You write a program to check if the contents of the file change, compared with the version you have locally:

```
import requests
import diggrtoolbox as dt

URL = 'https://diggr.github.io/platform_mapping/mediaartdb.json'
```

If the hashes turn out to be different, and you'd like to investigate the differences in more detail, we recommend using a diff-tool like dictdiffer.

1.3.6 deepget

The deepget function can be used easy with the results object of the TreeExplore search function, as demonstrated below:

```
>>> import diggrtoolbox as dt
>>> test_dict = {'id' : 123456789,
                 'data' : {'name' : 'diggr project',
                           'city' : 'Leipzig',
                           'field': 'Video Game Culture'},
                 'references':[{'url' : 'http://diggr.link',
                                'name' : 'diggr website'},
                               {'url' : 'http://ub.uni-leipzig.de',
                                'name' : 'UBL website'}]}
>>> tree = dt.TreeExplore(test_dict)
>>> results = tree.quiet_search("leipzig")
>>> for result in results:
        print(dt.deepget(test_dict, result['route']))
http://ub.uni-leipzig.de
```

The *TreeExplore* class itself also provides an easy method for accessing nested objects. Either a key, index, result dict or route can be used:

```
>>> print(tree[result])
http://ub.uni-leipzig.de
>>> print(tree[result['route']])
http://ub.uni-leipzig.de
>>> print(tree['references'][1]['url'])
http://ub.uni-leipzig.de
```

1.4 diggrtoolbox

1.4.1 diggrtoolbox package

Subpackages

diggrtoolbox.configgr package

Submodules

diggrtoolbox.configgr.configgr module

diggrtoolbox.schemaload package

Submodules

diggrtoolbox.schemaload.schemaload module

Module contents

diggrtoolbox.standardize package

Submodules

diggrtoolbox.standardize.standardize module

Module contents

diggrtoolbox.treeexplore package

Submodules

diggrtoolbox.treeexplore.treeexplore module

diggrtoolbox.treeexplore.treehash module

Module contents

diggrtoolbox.unified_api package

Submodules

diggrtoolbox.unified_api.diggr_api module

Module contents

diggrtoolbox.zipaccess package

Submodules

diggrtoolbox.zipaccess.zip_access module

Module contents

Module contents

- genindex
- search

CHAPTER 2

Authors, Copyright, License