Arcas Documentation

Release 0.0.1

Nikoleta Glynatsi

Apr 03, 2017

Contents

A large number of scholarly databases and collections offer some form of API access. An API is an online tool to access data straight from the databases.

Arcas is python tool designed to help communicate/ping various of these APIs.

CHAPTER 1

Table of Contents

Installing Arcas

From PyPi:

```
$ pip install arcas
```

From GitHub:

```
$ git clone https://github.com/Nikoleta-v3/Arcas.git
$ cd Arcas
$ pip install -r requirements.txt
$ python setup.py install
```

Arcas is supported by Python 3.5.

Tutorials

Arcas' tutorial wil cover the basic usage of the library. It covers a tutorial on retrieving a single article from a single API, retrieving the same article from various APIs and finally retrieving a large number of articles from different APIs.

Contents:

Tutorial I: Retrieving a single article

In this tutorial the aim is to retrieve a single article for the journal arXiv, where the word 'Game' is contained in the title or the abstract.

Initially, let us import Arcas:

```
>>> import arcas
```

The APIs, are implemented as classes. Here we make an API instance of the API arXiv:

```
>>> api = arcas.Arxiv()
```

We will now create the query, to which arXiv listens to. `records` is the number of records we are requesting for:

```
>>> parameters = api.parameters_fix(title='Game', abstract='Game', records=1)
>>> url = api.create_url_search(parameters)
```

The query will be used to ping the API and afterwards we parse the xml file that has been retrieved:

```
>>> request = api.make_request(url)
>>> root = api.get_root(request)
>>> raw_article = api.parse(root)
>>> article = api.to_dataframe(raw_article[0])
```

Note that we are using the library pandas to store the results. The data frame contains metadata on an article as they are recorded in the journal arXiv. Here we can type the following to see the columns of the data frame:

```
>>> article.columns
Index(['key', 'unique_key', 'title', 'author', 'abstract', 'date', 'journal',
       'pages', 'key_word', 'provenance'],dtype='object')
```

and we can ask for the title:

```
>>> article.title.unique()
    array([ 'A New Approach to Solve a Class of Continuous-Time Nonlinear
    Quadratic Zero-Sum Game Using ADP'], dtype=object)
```

The structure of the results is discussed in depth in *result set*.

Tutorial II: Retrieve an article from various APIs

In this tutorial we are aiming to make a similar query, to that in *tutorial I*, from different APIs.

To achieve that we will use a `for` loop, to loop over a list of given APIs classes. For each instance then repeat the following procedure:

```
>>>  for p in [arcas.Ieee, arcas.Plos, arcas.Arxiv, arcas.Springer, arcas.Nature]:

...      api = p()
...      parameters = api.parameters_fix(title='Game', abstract='Game', records=1)
...      url = api.create_url_search(parameters)
...      request = api.make_request(url)
...      root = api.get_root(request)
...      raw_article = api.parse(root)

...      for art in raw_article:
...          article = api.to_dataframe(art)
...          api.export(article, 'results_{}.json'.format(api.__class__.__name__))
```

The `export` function, is a function that writes the results to a json file. Here the results of each API are stored to a different file named after which API they come from.

Tutorial III: Retrieving a large number of articles

Now that we have learned to ping several APIs for a single article, we will repeat the procedure for a large number of articles. In this example the number of articles we would like to retrieve is 20 from each API.

Often, we are looking for hundreds of articles. Rather than asking the API for all the results at once, the APIs offer a paging mechanism through `start` and `records`. That way we can receive chunks of the result set at a time. `start` defines the index of the first returned article and `records` the number of articles returned by the query.

```
>>> for p in [arcas.Ieee, arcas.Plos, arcas.Arxiv, arcas.Springer, arcas.Nature]:
...     for start in range(2):
...
...         api = p()
...         parameters = api.parameters_fix(title='Game', abstract='Game',
...                                         records=10, start=(start * 10))
...         url = api.create_url_search(parameters)
...         request = api.make_request(url)
...         root = api.get_root(request)
...         raw_article = api.parse(root)
...
...     for art in raw_article:
...         article = api.to_dataframe(art)
...         api.export(article, 'results_{}.json'.format(api.__class__.__name__))
```

In our example this might not seem as an important difference. But assume you were asking for a hundred of articles. Some APIs have a limited number of articles that be can returned, thus using this practice we avoid overloading the API.

Reference

Contents:

List of available APIS

A list of the APIs you can ping with Arcas. Contents:

arXiv

arXiv Api is hosted at arXiv.org, is a document submission and retrieval system that is heavily used by the physics, mathematics and computer science communities.

arXiv is set as the default api for arcas. For more information visit the official site: https://arxiv.org/help/api/user-manual#Architecture.

IEEE Xplore

Query the Institute of Electrical and Electronics Engineers content repository and retrieve results for manipulation and presentation on local web interfaces.

For more information on IEEE Xplore visit the official site: http://ieeexplore.ieee.org/gateway/.

Nature

The nature.com OpenSearch API provides an open, bibliographic search service for content hosted on nature.com, comprising around half a million news and research articles and citations

For more information please visit the official site: http://www.nature.com/developers/documentation/api-references/opensearch-api/.

Springer

Springer Open Access API - Provides metadata and full-text content for more than 370,000 online documents from Springer open access xml, including BioMed Central and SpringerOpen journals.

Note that springer does not have an abstract search query and springer requires the user to register for a key. For more information visit the official site: https://dev.springer.com/restfuloperations.

PLOS

Query content from the seven open-access peer-reviewed journals from the Public Library of Science using any of the twenty-three terms in the PLOS Search.

For more information on PLOS Search API visit the official site: http://api.plos.org/solr/faq/.

Results set

Each response of the API returns a list of metadata for a given article. This list differs for each API. Arcas is designed to return a similar set of metadata for any given API. Thus the json results of Arcas has the following list of metadata:

- **key**
  - A generated key containing an authors name and publication year (e.g. Glynatsi2017)
- **unique_key**
  - A unique key generated using the hashlib python library. The hashable string is created by: [author name, title, year,abstract]
- **title**
  - Title of article
- **author**
  - A single entity of an author from the list of authors of the respective article
- **abstract**
  - The abstract of the article
- **date**
  - Date of publication
- **journal**
  - Journal of publication
- **pages**
  - Pages of publication
- **key_word**

  - A single entity of a keyword assigned to the article by the given journal

- **provenance**
  - Scholarly database for where the article was collected
- **score**
  - Score given to article by the given journal

CHAPTER 2

Indices and tables

- search