ads Documentation

Release 0.12.2

Andrew R. Casey

Mar 27, 2017

Contents

A Python Module to Interact with NASA's ADS that Doesn't Suck™

If you're in astro research, then you pretty much *need* NASA's ADS. It's tried, true, and people go crazy on the rare occasions when it goes down.

- Docs: https://ads.readthedocs.io/
- Repo: https://github.com/andycasey/ads
- PyPI: https://pypi.python.org/pypi/ads

CHAPTER 1

Getting Started

1. You'll need an API key from NASA ADS labs. Sign up for the newest version of ADS search at https://ui.adsabs.harvard.edu, visit account settings and generate a new API token. The official documentation is available at https://github.com/adsabs/adsabs-dev-api
2. When you get your API key, save it to a file called `~/.ads/dev_key` or save it as an environment variable named `ADS_DEV_KEY`
3. From a terminal type `pip install ads`

Happy Hacking!

CHAPTER 2

Examples

A list of available search fields is here: https://github.com/adsabs/adsabs-dev-api/blob/master/search.md#fields

You can use this module to search for some popular supernova papers:

```
>>> import ads
# Oops, I forgot to follow step 2 in "Getting Started"
>>> ads.config.token = 'my token'
>>> papers = ads.SearchQuery(q="supernova", sort="citation_count")
>>> for paper in papers:
>>>     print(paper.title)
[u'Maps of Dust Infrared Emission for Use in Estimation of Reddening and Cosmic␣
↪Microwave Background Radiation Foregrounds']
[u'Measurements of Omega and Lambda from 42 High-Redshift Supernovae']
[u'Observational Evidence from Supernovae for an Accelerating Universe and a␣
↪Cosmological Constant']
[u'First-Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Determination␣
↪of Cosmological Parameters']
[u'Abundances of the elements: Meteoritic and solar']
```

Or search for papers first-authored by someone:

```
>>> people = list(ads.SearchQuery(first_author="Reiss, A"))
>>> people[0].author
[u'Reiss, A. W.']
```

Or papers where they are anywhere in the author list:

```
>>> papers = list(ads.SearchQuery(author="Reiss, A"))
>>> papers[0].author
[u'Goodwin, F. E.', u'Henderson, D. M.', u'Reiss, A.', u'Wilkerson, John L.']
```

Or search by affiliation:

```
>>> papers = list(ads.SearchQuery(aff="*stromlo*"))
>>> papers[0].aff
```

```
[u'University of California, Berkeley',
 u'University of Kansas',
 u'Royal Greenwich Observatory',
 u"Queen's University",
 u'Mt. Stromlo Observatory',
 u'University of Durham']
```

In the above examples we *list()* the results from *ads.SearchQuery* because *ads.SearchQuery* is a generator, allowing us to return any number of articles. To prevent deep pagination of results, a default of *max_pages=3* is set. Feel free to change this, but be aware that each new page fetched will count against your daily API limit.

To retrieve information for articles where you know the ADS bibcodes:

```
>>> bibcodes = ['1994AAS...185.7506Z', '2001A&A...366...62A']
>>> articles = [list(ads.SearchQuery(bibcode=bibcode))[0] for bibcode in bibcodes]
```

Each object returned is an ``ads.Article`` object, which has a number of *very* handy attributes and functions:

```
>>> first_paper = papers[0]
>>> first_paper
<ads.search.Article at 0x7ff1b913dd10>
# Show some brief details about the paper
>>> print(first_paper)
<Zepf, S. et al. 1994, 1994AAS...185.7506Z>
# You can access attributes of an object in IPython by using the 'tab' button:
>>> first_paper.
first_paper.abstract              first_paper.build_citation_tree   first_paper.first_
↪author_norm     first_paper.keys                  first_paper.pubdate
first_paper.aff                   first_paper.build_reference_tree  first_paper.id    ␣
↪                 first_paper.keyword               first_paper.read_count
first_paper.author                first_paper.citation              first_paper.
↪identifier            first_paper.metrics               first_paper.reference
first_paper.bibcode               first_paper.citation_count        first_paper.issue␣
↪                 first_paper.page                  first_paper.title
first_paper.bibstem               first_paper.database              first_paper.items␣
↪                 first_paper.property              first_paper.volume
first_paper.bibtex                first_paper.first_author          first_paper.
↪iteritems             first_paper.pub                   first_paper.year
```

Which allows you to easily build complicated queries. Feel free to fork this repository and add your own examples!

CHAPTER 3

Rate limits and optimisations

Sandbox

The ADS's API uses rate limits to ensure that people cannot abuse it, but this can be pretty annoying if you get locked out when trying to build an application. To avoid such scenarios, you can use the ADS sandbox environment when prototyping:

```
>>> import ads.sandbox as ads
>>> q = ads.SearchQuery(q='star')
>>> for paper in q:
>>>     print(paper.title, paper.citation_count)
```

This will not access the live API, but provide mocked responses. When you're ready to go live with your code, simply change the import line to:

```
>>> import ads
```

and you're ready to go.

Rate limit usage

There are helpers that let you keep track of your rate limit, so you can also see how many requests your application is making to the API:

```
>>> import ads
>>> q = ads.SearchQuery(q='star')
>>> for paper in q:
>>>     print(paper.title, paper.citation_count)
>>> q.response.get_ratelimits()
{'limit': '5000', 'remaining': '4899', 'reset': '1459987200'}
```

or:

```
>>> import ads
>>> r = ads.RateLimits('SearchQuery')
>>> q = ads.SearchQuery(q='star')
>>> for paper in q:
>>>     print(paper.title, paper.citation_count)
>>> r.get_info()
{'SearchQuery': {'limit': '5000', 'remaining': '4899', 'reset': '1459987200'}}
```

If you prefer to use your own mocking package, or mock your responses manually, you can access both the stubdata and HTTPretty mocks from the package:

```
>>> import ads
>>> from ads.tests import mocks
>>>
>>> q = ads.SearchQuery(q='star')
>>> with mocks.MockSolrResponse(ads.SEARCH_URL):
>>>     q.execute()
>>> print(q.articles[0].title)
....
```

and:

```
>>> from ads.tests.stubdata import solr
>>> print(solr.example_solr_response)
```

CHAPTER 4

Lazy loading of attributes

One thing to be aware of when making queries is the use of lazy loading. This feature is great when prototyping some code, but can hurt you when in production. In the following example, *citation_count* is requested (for each paper) from the ADS API because it was not returned on the first request.

```
>>> import ads
>>> q = ads.SearchQuery(q='star')
>>> for paper in q:
>>>     print(paper.title, paper.citation_count)
```

This would result in *N=1+number_of_docs* requests rather than *N=1*. To ensure it is *N=1* you can request the field ahead of time:

```
>>> import ads
>>> q = ads.SearchQuery(q='star', fl=['id', 'bibcode', 'title', 'citation_count'])
>>> for paper in q:
>>>     print(paper.title, paper.citation_count)
```

CHAPTER 5

Authors

Vladimir Sudilovsky & Andy Casey, Geert Barentsen, Dan Foreman-Mackey, Miguel de Val-Borro, and Jonny Elliott

CHAPTER 6

License

This is open source software available under the MIT License. For details see the LICENSE file.