habanero Documentation

Release 0.6.0.3

Scott Chamberlain

May 24, 2018

Installation

Python client for the Crossref API

Source on GitHub at sckott/habanero

CHAPTER 1

Getting help

Having trouble? Or want to know how to get started?

- Try the *FAQ* – it's got answers to some common questions.
- Looking for specific information? Try the genindex
- Report bugs with habanero in our issue tracker.

CHAPTER 2

Installation

2.1 Installation guide

2.1.1 Installing habanero

Stable from pypi

```
pip install habanero
```

Development version

```
[sudo] pip install git+git://github.com/sckott/habanero.git#egg=habanero
```

Installation guide How to install habanero.

CHAPTER 3

Docs

3.1 Frequently Asked Questions

3.1.1 What other Crossref clients are out there?

- R: rcrossref
- Ruby: serrano
- Javascript: crossref

3.2 Usecases

3.2.1 Use case 1: Faceted search to get number of works per license type

Load library

```
from habanero import Crossref
cr = Crossref()
```

First, do a search like

```
res = cr.works(facet = "license:*")
```

Count number of unique licenses

```
res['message']['facets']['license']['value-count']
```

That's a lot of licenses!

Get licenses with > 1000 works

```
gt1000 = {k:v for (k,v) in res['message']['facets']['license']['values'].items() if v␣
↪> 1000}
len(gt1000)
```

Ah, that's only 63

Find the license with the most works

```
max(gt1000, key=lambda k: gt1000[k])
```

That's a license "http://www.elsevier.com/tdm/userlicense/1.0/" from Elsevier

Frequently Asked Questions Frequently asked questions.

Usecases Usecases for habanero.

CHAPTER 4

Modules

4.1 habanero modules

habanero is split up into modules.

- Crossref - Core Crossref APIs for search, journals, members, etc.
- Counts - Crossref citation counts
- Content negotation - Content negotation

You can import the entire library, or each module individually as needed.

4.2 crossref module

crossref module API:

- *works*
- *members*
- *prefixes*
- *funders*
- *journals*
- *types*
- *licenses*
- *registration_agency*
- *random_dois*

Example usage:

```
from habanero import Crossref
cr = Crossref()
cr.works()
cr.works(ids = '10.1371/journal.pone.0033693')
cr.works(query = "ecology")
```

4.2.1 crossref API

Crossref.**works**(*ids=None*, *query=None*, *filter=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, *select=None*, *cursor=None*, *cursor_max=5000*, ***kwargs*)

Search Crossref works

Parameters

- **ids** – [Array] DOIs (digital object identifier) or other identifiers
- **query** – [String] A query string
- **filter** – [Hash] Filter options. See examples for usage. Accepts a dict, with filter names and their values. For repeating filter names pass in a list of the values to that filter name, e.g., *{'award_funder': ['10.13039/100004440', '10.13039/100000861']}*. See https://github.com/CrossRef/rest-api-doc#filter-names for filter names and their descriptions and `filter_names()` and `filter_details()`
- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sample** – [Fixnum] Number of random results to return. when you use the sample parameter, the limit and offset parameters are ignored. Max: 100
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See **sorting_** for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=**. See **Facets_** for options.
- **select** – [String/list(Strings)] Crossref metadata records can be quite large. Sometimes you just want a few elements from the schema. You can "select" a subset of elements to return. This can make your API calls much more efficient. Not clear yet which fields are allowed here.
- **cursor** – [String] Cursor character string to do deep paging. Default is None. Pass in '*' to start deep paging. Any combination of query, filters and facets may be used with deep paging cursors. While rows may be specified along with cursor, offset and sample cannot be used. See https://github.com/CrossRef/rest-api-doc/blob/master/rest_api.md#deep-paging-with-cursors
- **cursor_max** – [Fixnum] Max records to retrieve. Only used when cursor param used. Because deep paging can result in continuous requests until all are retrieved, use this parameter to set a maximum number of records. Of course, if there are less records found than this value, you will get only those found.
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and **FieldQueries_**)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.works()
cr.works(ids = '10.1371/journal.pone.0033693')
dois = ['10.1371/journal.pone.0033693', ]
cr.works(ids = dois)
x = cr.works(query = "ecology")
x['status']
x['message-type']
x['message-version']
x['message']
x['message']['total-results']
x['message']['items-per-page']
x['message']['query']
x['message']['items']

# Get full text links
x = cr.works(filter = {'has_full_text': True})
x

# Parse output to various data pieces
x = cr.works(filter = {'has_full_text': True})
## get doi for each item
[ z['DOI'] for z in x['message']['items'] ]
## get doi and url for each item
[ {"doi": z['DOI'], "url": z['URL']} for z in x['message']['items'] ]
### print every doi
for i in x['message']['items']:
     print i['DOI']

# filters - pass in as a dict
## see https://github.com/CrossRef/rest-api-doc#filter-names
cr.works(filter = {'has_full_text': True})
cr.works(filter = {'has_funder': True, 'has_full_text': True})
cr.works(filter = {'award_number': 'CBET-0756451', 'award_funder': '10.13039/
↪100000001'})
## to repeat a filter name, pass in a list
x = cr.works(filter = {'award_funder': ['10.13039/100004440', '10.13039/100000861
↪']}, limit = 100)
map(lambda z:z['funder'][0]['DOI'], x['message']['items'])

# Deep paging, using the cursor parameter
## this search should lead to only ~215 results
cr.works(query = "widget", cursor = "*", cursor_max = 100)
## this search should lead to only ~2500 results, in chunks of 500
res = cr.works(query = "octopus", cursor = "*", limit = 500)
sum([ len(z['message']['items']) for z in res ])
## about 167 results
res = cr.works(query = "extravagant", cursor = "*", limit = 50, cursor_max = 500)
sum([ len(z['message']['items']) for z in res ])
## cursor_max to get back only a maximum set of results
res = cr.works(query = "widget", cursor = "*", cursor_max = 100)
sum([ len(z['message']['items']) for z in res ])
## cursor_max - especially useful when a request could be very large
```

(continues on next page)

(continued from previous page)

```
### e.g., "ecology" results in ~275K records, lets max at 10,000
###   with 1000 at a time
res = cr.works(query = "ecology", cursor = "*", cursor_max = 10000, limit = 1000)
sum([ len(z['message']['items']) for z in res ])
items = [ z['message']['items'] for z in res ]
items = [ item for sublist in items for item in sublist ]
[ z['DOI'] for z in items ][0:50]

# field queries
res = cr.works(query = "ecology", query_author = 'carl boettiger')
[ x['author'][0]['family'] for x in res['message']['items'] ]

# select certain fields to return
## as a comma separated string
cr.works(query = "ecology", select = "DOI,title")
## or as a list
cr.works(query = "ecology", select = ["DOI","title"])
```

Crossref.**members**(*ids=None*, *query=None*, *filter=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, *works=False*, *select=None*, *cursor=None*, *cursor_max=5000*, ***kwargs*)

Search Crossref members

Parameters

- **ids** – [Array] DOIs (digital object identifier) or other identifiers
- **query** – [String] A query string
- **filter** – [Hash] Filter options. See examples for usage. Accepts a dict, with filter names and their values. For repeating filter names pass in a list of the values to that filter name, e.g., *{'award_funder': ['10.13039/100004440', '10.13039/100000861']}*. See https://github.com/CrossRef/rest-api-doc#filter-names for filter names and their descriptions and `filter_names()` and `filter_details()`
- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sample** – [Fixnum] Number of random results to return. when you use the sample parameter, the limit and offset parameters are ignored. This parameter only used when works requested. Max: 100
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See **sorting_** for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=** See **Facets_** for options.
- **select** – [String/list(Strings)] Crossref metadata records can be quite large. Sometimes you just want a few elements from the schema. You can "select" a subset of elements to return. This can make your API calls much more efficient. Not clear yet which fields are allowed here.
- **works** – [Boolean] If true, works returned as well. Default: false

- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and FieldQueries_)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.members(ids = 98)

# get works
res = cr.members(ids = 98, works = True, limit = 3)
len(res['message']['items'])
[ z['DOI'] for z in res['message']['items'] ]

# cursor - deep paging
res = cr.members(ids = 98, works = True, cursor = "*")
sum([ len(z['message']['items']) for z in res ])
items = [ z['message']['items'] for z in res ]
items = [ item for sublist in items for item in sublist ]
[ z['DOI'] for z in items ][0:50]

# field queries
res = cr.members(ids = 98, works = True, query_author = 'carl boettiger', limit =␣
↪7)
[ x['author'][0]['family'] for x in res['message']['items'] ]
```

Crossref.**prefixes**(*ids=None*, *filter=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, *works=False*, *select=None*, *cursor=None*, *cursor_max=5000*, ***kwargs*)

Search Crossref prefixes

Parameters

- **ids** – [Array] DOIs (digital object identifier) or other identifiers
- **filter** – [Hash] Filter options. See examples for usage. Accepts a dict, with filter names and their values. For repeating filter names pass in a list of the values to that filter name, e.g., *{'award_funder': ['10.13039/100004440', '10.13039/100000861']}*. See https://github.com/CrossRef/rest-api-doc#filter-names for filter names and their descriptions and `filter_names()` and `filter_details()`
- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sample** – [Fixnum] Number of random results to return. when you use the sample parameter, the limit and offset parameters are ignored. This parameter only used when works requested. Max: 100
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See sorting_ for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=** See Facets_ for options.

- **select** – [String/list(Strings)] Crossref metadata records can be quite large. Sometimes you just want a few elements from the schema. You can "select" a subset of elements to return. This can make your API calls much more efficient. Not clear yet which fields are allowed here.
- **works** – [Boolean] If true, works returned as well. Default: false
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and **FieldQueries_**)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.prefixes(ids = "10.1016")
cr.prefixes(ids = ['10.1016','10.1371','10.1023','10.4176','10.1093'])

# get works
cr.prefixes(ids = "10.1016", works = True)

# Limit number of results
cr.prefixes(ids = "10.1016", works = True, limit = 3)

# Sort and order
cr.prefixes(ids = "10.1016", works = True, sort = "relevance", order = "asc")

# cursor - deep paging
res = cr.prefixes(ids = "10.1016", works = True, cursor = "*", limit = 200)
sum([ len(z['message']['items']) for z in res ])
items = [ z['message']['items'] for z in res ]
items = [ item for sublist in items for item in sublist ]
[ z['DOI'] for z in items ][0:50]

# field queries
res = cr.prefixes(ids = "10.1371", works = True, query_editor = 'cooper', filter
↪= {'type': 'journal-article'})
eds = [ x.get('editor') for x in res['message']['items'] ]
[ z for z in eds if z is not None ]
```

Crossref.**funders**(*ids=None*, *query=None*, *filter=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, *works=False*, *select=None*, *cursor=None*, *cursor_max=5000*, ***kwargs*)

Search Crossref funders

Note that funders without IDs don't show up on the */funders* route, that is, won't show up in searches via this method

Parameters

- **ids** – [Array] DOIs (digital object identifier) or other identifiers
- **query** – [String] A query string
- **filter** – [Hash] Filter options. See examples for usage. Accepts a dict, with filter names and their values. For repeating filter names pass in a list of the values to that filter name, e.g., *{'award_funder': ['10.13039/100004440', '10.13039/100000861']}*. See https://github.com/CrossRef/rest-api-doc#filter-names for filter names and their descriptions and `filter_names()` and `filter_details()`

- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sample** – [Fixnum] Number of random results to return. when you use the sample parameter, the limit and offset parameters are ignored. This parameter only used when works requested. Max: 100
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See **sorting_** for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=** See **Facets_** for options.
- **select** – [String/list(Strings)] Crossref metadata records can be quite large. Sometimes you just want a few elements from the schema. You can "select" a subset of elements to return. This can make your API calls much more efficient. Not clear yet which fields are allowed here.
- **works** – [Boolean] If true, works returned as well. Default: false
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and **FieldQueries_**)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.funders(ids = '10.13039/100000001')
cr.funders(query = "NSF")

# get works
cr.funders(ids = '10.13039/100000001', works = True)

# cursor - deep paging
res = cr.funders(ids = '10.13039/100000001', works = True, cursor = "*", limit =␣
↪200)
sum([ len(z['message']['items']) for z in res ])
items = [ z['message']['items'] for z in res ]
items = [ item for sublist in items for item in sublist ]
[ z['DOI'] for z in items ][0:50]

# field queries
res = cr.funders(ids = "10.13039/100000001", works = True, query_container_title␣
↪= 'engineering', filter = {'type': 'journal-article'})
eds = [ x.get('editor') for x in res['message']['items'] ]
[ z for z in eds if z is not None ]
```

Crossref.**journals**(*ids=None*, *query=None*, *filter=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, *works=False*, *select=None*, *cursor=None*, *cursor_max=5000*, ***kwargs*)

Search Crossref journals

Parameters

- **ids** – [Array] DOIs (digital object identifier) or other identifiers

- **query** – [String] A query string
- **filter** – [Hash] Filter options. See examples for usage. Accepts a dict, with filter names and their values. For repeating filter names pass in a list of the values to that filter name, e.g., *{'award_funder': ['10.13039/100004440', '10.13039/100000861']}*. See https://github.com/CrossRef/rest-api-doc#filter-names for filter names and their descriptions and `filter_names()` and `filter_details()`
- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sample** – [Fixnum] Number of random results to return. when you use the sample parameter, the limit and offset parameters are ignored. This parameter only used when works requested. Max: 100
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See **sorting_** for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=**. See **Facets_** for options.
- **select** – [String/list(Strings)] Crossref metadata records can be quite large. Sometimes you just want a few elements from the schema. You can "select" a subset of elements to return. This can make your API calls much more efficient. Not clear yet which fields are allowed here.
- **works** – [Boolean] If true, works returned as well. Default: false
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and **FieldQueries_**)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.journals(ids = "2167-8359")
cr.journals()

# pass many ids
cr.journals(ids = ['1803-2427', '2326-4225'])

# search
cr.journals(query = "ecology")
cr.journals(query = "peerj")

# get works
cr.journals(ids = "2167-8359", works = True)
cr.journals(ids = "2167-8359", query = 'ecology', works = True, sort = 'score',␣
↪order = "asc")
cr.journals(ids = "2167-8359", query = 'ecology', works = True, sort = 'score',␣
↪order = "desc")
cr.journals(ids = "2167-8359", works = True, filter = {'from_pub_date': '2014-03-
↪03'})
cr.journals(ids = '1803-2427', works = True)
```

(continues on next page)

(continued from previous page)

```
cr.journals(ids = '1803-2427', works = True, sample = 1)
cr.journals(limit: 2)

# cursor - deep paging
res = cr.funders(ids = '10.13039/100000001', works = True, cursor = "*", limit =␣
↪200)
sum([ len(z['message']['items']) for z in res ])
items = [ z['message']['items'] for z in res ]
items = [ item for sublist in items for item in sublist ]
[ z['DOI'] for z in items ][0:50]

# field queries
res = cr.journals(ids = "2167-8359", works = True, query_title = 'fish', filter =
↪{'type': 'journal-article'})
[ x.get('title') for x in res['message']['items'] ]
```

Crossref.**types**(*ids=None*, *query=None*, *filter=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, *works=False*, *select=None*, *cursor=None*, *cursor_max=5000*, ***kwargs*)

Search Crossref types

Parameters

- **ids** – [Array] Type identifier, e.g., journal
- **query** – [String] A query string
- **filter** – [Hash] Filter options. See examples for usage. Accepts a dict, with filter names and their values. For repeating filter names pass in a list of the values to that filter name, e.g., *{'award_funder': ['10.13039/100004440', '10.13039/100000861']}*. See https://github.com/CrossRef/rest-api-doc#filter-names for filter names and their descriptions and `filter_names()` and `filter_details()`
- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sample** – [Fixnum] Number of random results to return. when you use the sample parameter, the limit and offset parameters are ignored. This parameter only used when works requested. Max: 100
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See **sorting_** for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=** See **Facets_** for options.
- **select** – [String/list(Strings)] Crossref metadata records can be quite large. Sometimes you just want a few elements from the schema. You can "select" a subset of elements to return. This can make your API calls much more efficient. Not clear yet which fields are allowed here.
- **works** – [Boolean] If true, works returned as well. Default: false
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and **FieldQueries_**)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.types()
cr.types(ids = "journal")
cr.types(ids = "journal-article")
cr.types(ids = "journal", works = True)

# field queries
res = cr.types(ids = "journal-article", works = True, query_title = 'gender',␣
↪rows = 100)
[ x.get('title') for x in res['message']['items'] ]
```

Crossref.**licenses**(*query=None*, *offset=None*, *limit=None*, *sample=None*, *sort=None*, *order=None*, *facet=None*, ***kwargs*)

Search Crossref licenses

Parameters

- **query** – [String] A query string
- **offset** – [Fixnum] Number of record to start at, from 1 to 10000
- **limit** – [Fixnum] Number of results to return. Not relavant when searching with specific dois. Default: 20. Max: 1000
- **sort** – [String] Field to sort on. Note: If the API call includes a query, then the sort order will be by the relevance score. If no query is included, then the sort order will be by DOI update date. See **sorting_** for possible values.
- **order** – [String] Sort order, one of 'asc' or 'desc'
- **facet** – [Boolean/String] Set to *true* to include facet results (default: false). Optionally, pass a query string, e.g., *facet=type-name:** or *facet=license=** See **Facets_** for options.
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples and **FieldQueries_**)

Returns A dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.licenses()
cr.licenses(query = "creative")
```

Crossref.**registration_agency**(*ids*, ***kwargs*)

Determine registration agency for DOIs

Parameters

- **ids** – [Array] DOIs (digital object identifier) or other identifiers
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples)

Returns list of DOI minting agencies

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.registration_agency('10.1371/journal.pone.0033693')
cr.registration_agency(ids = ['10.1007/12080.1874-1746','10.1007/10452.1573-5125',
↪ '10.1111/(issn)1442-9993'])
```

Crossref.**random_dois**(*sample=10*, ***kwargs*)

Get a random set of DOIs

Parameters

- **sample** – [Fixnum] Number of random DOIs to return. Default: 10. Max: 100
- **kwargs** – additional named arguments passed on to *requests.get*, e.g., field queries (see examples)

Returns [Array] of DOIs

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.random_dois(1)
cr.random_dois(10)
cr.random_dois(50)
cr.random_dois(100)
```

4.3 Crossref Search Filters

crossref module API:

- *filter_names*
- *filter_details*

Example usage:

```
from habanero import Crossref
cr = Crossref()
cr.filter_names()
cr.filter_details()
```

4.3.1 filters API

static Crossref.**filter_names**()

Filter names - just the names of each filter

Filters are used in the Crossref search API to modify searches

Returns dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.filter_names()
```

static Crossref.**filter_details**()
Filter details - filter names, possible values, and description

Filters are used in the Crossref search API to modify searches

Returns dict

Usage:

```
from habanero import Crossref
cr = Crossref()
cr.filter_details()
# Get descriptions for each filter
x = cr.filter_details()
[ z['description'] for z in x.values() ]
```

4.4 counts module

counts module API:

- *citation_count*

Example usage:

```
from habanero import counts
counts.citation_count(doi = "10.1371/journal.pone.0042793")
counts.citation_count(doi = "10.1016/j.fbr.2012.01.001")
```

4.4.1 counts API

counts.**citation_count**(*url='http://www.crossref.org/openurl/'*, *key='cboettig@ropensci.org'*, ***kwargs*)
Get a citation count with a DOI

Parameters

- **doi** – [String] DOI, digital object identifier
- **url** – [String] the API url for the function (should be left to default)
- **keyc** – [String] your API key

See http://labs.crossref.org/openurl/ for more info on this Crossref API service.

Usage:

```
from habanero import counts
counts.citation_count(doi = "10.1371/journal.pone.0042793")
counts.citation_count(doi = "10.1016/j.fbr.2012.01.001")
# DOI not found
## FIXME
counts.citation_count(doi = "10.1016/j.fbr.2012")
```

4.5 cn module

cn module API:

- *content_negotiation*

Example usage:

```
from habanero import cn
cn.content_negotiation(ids = '10.1126/science.169.3946.635')
cn.content_negotiation(ids = '10.1126/science.169.3946.635', format = "citeproc-json")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "rdf-xml")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "crossref-xml")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "bibentry")
```

4.5.1 cn API

cn.**content_negotiation**(*format='bibtex'*, *style='apa'*, *locale='en-US'*, *url=None*, ***kwargs*)

Get citations in various formats from CrossRef

Parameters

- **ids** – [str] Search by a single DOI or many DOIs, each a string. If many passed in, do so in a list
- **format** – [str] Name of the format. One of "rdf-xml", "turtle", "citeproc-json", "citeproc-json-ish", "text", "ris", "bibtex" (Default), "crossref-xml", "datacite-xml","bibentry", or "crossref-tdm"
- **style** – [str] A CSL style (for text format only). See `csl_styles()` for options. Default: "apa". If there's a style that CrossRef doesn't support you'll get a *(500) Internal Server Error*
- **locale** – [str] Language locale. See *locale.locale_alias*
- **url** – [str] Base URL for the content negotiation request. Default: *https://doi.org*
- **kwargs** – any additional arguments will be passed on to *requests.get*

Returns string, which can be parsed to various formats depending on what format you request (e.g., JSON vs. XML vs. bibtex)

Usage:

```
from habanero import cn
cn.content_negotiation(ids = '10.1126/science.169.3946.635')

# get citeproc-json
cn.content_negotiation(ids = '10.1126/science.169.3946.635', format = "citeproc-
↪json")

# some other formats
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "rdf-xml")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "crossref-
↪xml")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text")

# return an R bibentry type
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "bibentry")
cn.content_negotiation(ids = "10.6084/m9.figshare.97218", format = "bibentry")

# return an apa style citation
```

(continues on next page)

(continued from previous page)

```
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text",␣
↪style = "apa")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text",␣
↪style = "harvard3")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text",␣
↪style = "elsevier-harvard")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text",␣
↪style = "ecoscience")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text",␣
↪style = "heredity")
cn.content_negotiation(ids = "10.1126/science.169.3946.635", format = "text",␣
↪style = "oikos")

# Using DataCite DOIs
## some formats don't work
# cn.content_negotiation(ids = "10.5284/1011335", format = "text")
# cn.content_negotiation(ids = "10.5284/1011335", format = "crossref-xml")
# cn.content_negotiation(ids = "10.5284/1011335", format = "crossref-tdm")

## But most do work
cn.content_negotiation(ids = "10.5284/1011335", format = "datacite-xml")
cn.content_negotiation(ids = "10.5284/1011335", format = "rdf-xml")
cn.content_negotiation(ids = "10.5284/1011335", format = "turtle")
cn.content_negotiation(ids = "10.5284/1011335", format = "citeproc-json")
cn.content_negotiation(ids = "10.5284/1011335", format = "ris")
cn.content_negotiation(ids = "10.5284/1011335", format = "bibtex")
cn.content_negotiation(ids = "10.5284/1011335", format = "bibentry")
cn.content_negotiation(ids = "10.5284/1011335", format = "bibtex")

# many DOIs
dois = ['10.5167/UZH-30455','10.5167/UZH-49216','10.5167/UZH-503', '10.5167/UZH-
↪38402','10.5167/UZH-41217']
x = cn.content_negotiation(ids = dois)

# Use a different base url
url = "http://dx.doi.org"
cn.content_negotiation(ids = "10.1126/science.169.3946.635", url = url)
cn.content_negotiation(ids = "10.5284/1011335", url = url)
```

habanero modules Introduction to habanero modules.

crossref module The crossref module: core Crossref APIs.

Crossref Search Filters The filters module: Filters details for use with Crossref module.

counts module The counts module: Crossref citation counts.

cn module The cn module: Crossref content negotation.

CHAPTER 5

All the rest

5.1 Exceptions

class habanero.**RequestError**
Exception raised for request errors.

This error occurrs when a request sent to the Crossref API results in an error. We give back:

- HTTP status code
- Error message

5.2 Changelog

5.2.1 0.6.0 (2017-10-20)

- Added verification and docs for additional Crossref search filters (#62)
- Big improvement to docs on readthedocs (#59)
- Added *mailto* support (#68) (#63) and related added docs about polite pool (#66)
- Added support for *select* parameter (#65)
- Added all new */works* route filters, and simplified filter option handling within library (#60)

5.2.2 0.5.0 (2017-07-20)

- Now using *vcrpy* to mock all unit tests (#54)
- Can now set your own base URL for content negotation (#37)
- Some field queries with *works()* were failing, but now seem to be working, likely due to fixes in Crossref API (#53)

- style input to *content_negotiation* was fixed (#57) (#58) thanks @talbertc-usgs
- Fix to *content_negotiation* when inputting a DOI as a unicode string (#56)

5.2.3 0.3.0 (2017-05-21)

- Added more documentation for field queries, describing available fields that support field queries, and how to do field queries (#50)
- *sample* parameter maximum value is 100 - has been for a while, but wasn't updated in Crossref docs (#44)
- Updated docs that *facet* parameter can be a string query in addition to a boolean (#49)
- Documented new 10,000 max value for */works* requests - that is, for the *offset* parameter - if you need more results than that use *cursor* (see https://github.com/CrossRef/rest-api-doc/blob/master/rest_api.md#deep-paging-with-cursors) (#47)
- Added to docs a bit about rate limiting, their current values, that they can change, and how to show them in verbose curl responses (#45)
- Now using *https://doi.org* for *cn.content_negotation* - and function gains new parameter *url* to specify different base URLs for content negotiation (#36)
- Fixes to kwargs and fix docs for what can be passed to kwargs (#41)
- Duplicated names passed to *filter* were not working - fixed now (#48)
- Raise proper HTTP errors when appropriate for *cn.content_negotiation* thanks @jmaupetit (#55)

5.2.4 0.2.6 (2016-06-24)

- fixed problem with *cr.works()* where DOIs passed weren't making the correct API request to Crossref (#40)
- added support for field queries to all methods that support */works* (<https://github.com/CrossRef/rest-api-doc/blob/master/rest_api.md#field-queries>) (#38)

5.2.5 0.2.2 (2016-03-09)

- fixed some example code that included non-working examples (#34)
- fixed bug in *registration_agency()* method, works now! (#35)
- removed redundant *filter_names* and *filter_details* bits in docs

5.2.6 0.2.0 (2016-02-10)

- user-agent strings now passed in every http request to Crossref, including a *X-USER-AGENT* header in case the *User-Agent* string is lost (#33)
- added a disclaimer to docs about what is actually searched when searching the Crossref API - that is, only what is returned in the API, so no full text or abstracts are searched (#32)
- improved http error parsing - now passes on the hopefully meaningful error messages from the Crossref API (#31)
- more tests added (#30)
- habanero now supports cursor for deep paging. note that cursor only works with requests to the */works* route (#18)

5.2.7 0.1.3 (2015-12-02)

- Fix wheel file to be a universal to install on python2 and python3 (#25)
- Added method *csl_styles* to get CSL styles for use in content negotiation (#27)
- More documentation for content negotiation (#26)
- Made note in docs that *sample* param ignored unless */works* used (#24)
- Made note in docs that funders without IDs don't show up on the */funders* route (#23)

5.2.8 0.1.1 (2015-11-17)

- Fix readme

5.2.9 0.1.0 (2015-11-17)

- Now compatible with Python 2x and 3x
- *agency()* method changed to *registration_agency()*
- New method *citation_count()* - get citation counts for DOIs
- New method *crosscite()* - get a citation for DOIs, only supports simple text format
- New method *random_dois()* - get a random set of DOIs
- Now importing *xml.dom* to do small amount of XML parsing
- Changed library structure, now with module system, separated into modules for the main Crossref search API (i.e., *api.crossref.org*) including higher level methods (e.g., *registration_agency*), content negotiation, and citation counts.

5.2.10 0.0.6 (2015-11-09)

- First pypi release

5.3 Contributors

- Scott Chamberlain
- Julien Maupetit
- Steve Peak
- Colin Talbert
- Daniel Himmelstein
- Kyle Niemeyer
- ioverka

5.4 Contributing

Important: Double check you are reading the most recent version of this document at http://habanero.readthedocs.io/en/latest/index.html

5.4.1 Bug reports

Please report bug reports on our issue tracker.

5.4.2 Feature requests

Please put feature requests on our issue tracker.

5.4.3 Pull requests

When you submit a PR you'll see a template that pops up - it's reproduced here.

- Provide a general summary of your changes in the Title
- Describe your changes in detail
- If the PR closes an issue make sure include e.g., *fix #4* or similar, or if just relates to an issue make sure to mention it like *#4*
- If introducing a new feature or changing behavior of existing methods/functions, include an example if possible to do in brief form
- Did you remember to include tests? Unless you're changing docs/grammar, please include new tests for your change

5.4.4 Writing tests

We're using *nosetests* for testing. See the nosetests docs for help on contributing to or writing tests.

The Makefile has tasks for testing under python 2 and 3

```
make test
make test3
```

5.5 Contributor Code of Conduct

As contributors and maintainers of this project, we pledge to respect all people who contribute through reporting issues, posting feature requests, updating documentation, submitting pull requests or patches, and other activities.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, or religion.

Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct. Project maintainers who do not follow the Code of Conduct may be removed from the project team.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by opening an issue or contacting one or more of the project maintainers.

This Code of Conduct is adapted from the Contributor Covenant (http:contributor-covenant.org), version 1.0.0, available at http://contributor-covenant.org/version/1/0/0/

5.6 LICENSE

5.7 Indices and tables

- genindex
- modindex
- search

Python Module Index

h

habanero, 19

Index