pygbif Documentation

Release 0.1.4

Scott Chamberlain

Jun 21, 2017

Contents

Python client for the GBIF API.

Source on GitHub at sckott/pygbif

pygbif is split up into modules for each of the major groups of API methods.

- Registry - Datasets, Nodes, Installations, Networks, Organizations - Registry API Docs
- Species - Taxonomic names - Species API Docs
- Occurrences - Occurrence data, including the download API - Occurrences API Docs

Note that GBIF maps API is not included in *pygbif*.

Other GBIF clients:

- R: rgbif

CHAPTER 1

Installation

```
pip install pygbif
```

CHAPTER 2

Registry module

```
from pygbif import registry
registry.dataset_metrics(uuid='3f8a1297-3259-4700-91fc-acc4170b27ce')
```

CHAPTER 3

Species module

```
from pygbif import species
species.name_suggest(q='Puma concolor')
```

CHAPTER 4

Occurrences module

```python
from pygbif import occurrences as occ
occ.search(taxonKey = 3329049)
occ.get(key = 252408386)
occ.count(isGeoreferenced = True)
occ.download_list(user = "sckott", limit = 5)
occ.download_meta(key = "0000099-140929101555934")
occ.download_get("0000066-140928181241064")
```

CHAPTER 5

Meta

- License: MIT, see LICENSE file
- Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

Contents

occurrences module

`occurrences.search`(*taxonKey=None*, *scientificName=None*, *country=None*, *publishingCountry=None*, *hasCoordinate=None*, *typeStatus=None*, *recordNumber=None*, *lastInterpreted=None*, *continent=None*, *geometry=None*, *recordedBy=None*, *basisOfRecord=None*, *datasetKey=None*, *eventDate=None*, *catalogNumber=None*, *year=None*, *month=None*, *decimalLatitude=None*, *decimalLongitude=None*, *elevation=None*, *depth=None*, *institutionCode=None*, *collectionCode=None*, *hasGeospatialIssue=None*, *issue=None*, *q=None*, *mediatype=None*, *limit=300*, *offset=0*, ***kwargs*)

Search GBIF occurrences

Parameters

- **`taxonKey`** – [int] A GBIF occurrence identifier
- **`scientificName`** – [str] A scientific name from the GBIF backbone. All included and synonym taxa are included in the search.
- **`datasetKey`** – [str] The occurrence dataset key (a uuid)
- **`catalogNumber`** – [str] An identifier of any form assigned by the source within a physical collection or digital dataset for the record which may not unique, but should be fairly unique in combination with the institution and collection code.
- **`recordedBy`** – [str] The person who recorded the occurrence.

- **collectionCode** – [str] An identifier of any form assigned by the source to identify the physical collection or digital dataset uniquely within the text of an institution.
- **institutionCode** – [str] An identifier of any form assigned by the source to identify the institution the record belongs to. Not guaranteed to be que.
- **country** – [str] The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded. See here http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2}
- **basisOfRecord** – [str] Basis of record, as defined in our BasisOfRecord enum here http://gbif.github.io/gbif-api/apidocs/org/gbif/api/vocabulary/BasisOfRecord.html Acceptable values are:
  - FOSSIL_SPECIMEN An occurrence record describing a fossilized specimen.
  - HUMAN_OBSERVATION An occurrence record describing an observation made by one or more people.
  - LITERATURE An occurrence record based on literature alone.
  - LIVING_SPECIMEN An occurrence record describing a living specimen, e.g.
  - MACHINE_OBSERVATION An occurrence record describing an observation made by a machine.
  - OBSERVATION An occurrence record describing an observation.
  - PRESERVED_SPECIMEN An occurrence record describing a preserved specimen.
  - UNKNOWN Unknown basis for the record.
- **eventDate** – [date] Occurrence date in ISO 8601 format: yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Supports range queries, smaller,larger (e.g., '1990,1991', whereas '1991,1990' wouldn't work)
- **year** – [int] The 4 digit year. A year of 98 will be interpreted as AD 98. Supports range queries, smaller,larger (e.g., '1990,1991', whereas '1991,1990' wouldn't work)
- **month** – [int] The month of the year, starting with 1 for January. Supports range queries, smaller,larger (e.g., '1,2', whereas '2,1' wouldn't work)
- **q** – [str] Query terms. The value for this parameter can be a simple word or a phrase.
- **decimalLatitude** – [float] Latitude in decimals between -90 and 90 based on WGS 84. Supports range queries, smaller,larger (e.g., '25,30', whereas '30,25' wouldn't work)
- **decimalLongitude** – [float] Longitude in decimals between -180 and 180 based on WGS 84. Supports range queries (e.g., '-0.4,-0.2', whereas '-0.2,-0.4' wouldn't work).
- **publishingCountry** – [str] The 2-letter country code (as per ISO-3166-1) of the country in which the occurrence was recorded.
- **elevation** – [int/str] Elevation in meters above sea level. Supports range queries, smaller,larger (e.g., '5,30', whereas '30,5' wouldn't work)
- **depth** – [int/str] Depth in meters relative to elevation. For example 10 meters below a lake surface with given elevation. Supports range queries, smaller,larger (e.g., '5,30', whereas '30,5' wouldn't work)
- **geometry** – [str] Searches for occurrences inside a polygon described in Well Known Text (WKT) format. A WKT shape written as either POINT, LINESTRING, LINEARRING or POLYGON. Example of a polygon: ((30.1 10.1, 20, 20 40, 40 40, 30.1 10.1)) would be queried as http://bit.ly/1BzNwDq}.

- **hasGeospatialIssue** – [bool] Includes/excludes occurrence records which contain spatial issues (as determined in our record interpretation), i.e. code{hasGeospatialIssue=TRUE} returns only those records with spatial issues while code{hasGeospatialIssue=FALSE} includes only records without spatial issues. The absence of this parameter returns any record with or without spatial issues.
- **issue** – [str] One or more of many possible issues with each occurrence record. See Details. Issues passed to this parameter filter results by the issue.
- **hasCoordinate** – (logical) Return only occurence records with lat/long data (TRUE) or all records (FALSE, default).
- **typeStatus** – [str] Type status of the specimen. One of many options. See ?typestatus
- **recordNumber** – [int] Number recorded by collector of the data, different from GBIF record number. See http://rs.tdwg.org/dwc/terms/#recordNumber} for more info
- **lastInterpreted** – [date] Date the record was last modified in GBIF, in ISO 8601 format: yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Supports range queries, smaller,larger (e.g., '1990,1991', whereas '1991,1990' wouldn't work)
- **continent** – [str] Continent. One of africa, antarctica, asia, europe, north_america (North America includes the Caribbean and reachies down and includes Panama), oceania, or south_america
- **fields** – [str] Default ('all') returns all fields. 'minimal' returns just taxon name, key, latitude, and longitute. Or specify each field you want returned by name, e.g. fields = c('name','latitude','elevation').
- **mediatype** – [str] Media type. Default is NULL, so no filtering on mediatype. Options: NULL, 'MovingImage', 'Sound', and 'StillImage'
- **limit** – [int] Number of results to return. Default: 300
- **offset** – [int] Record to start at. Default: 0

Returns A dictionary, of results

Usage:

```
from pygbif import occurrences
occurrences.search(taxonKey = 3329049)

# Return 2 results, this is the default by the way
occurrences.search(taxonKey=3329049, limit=2)

# Instead of getting a taxon key first, you can search for a name directly
## However, note that using this approach (with \code{scientificName="..."})
## you are getting synonyms too. The results for using \code{scientifcName} and
## \code{taxonKey} parameters are the same in this case, but I wouldn't be␣
↪surprised if for some
## names they return different results
occurrences.search(scientificName = 'Ursus americanus')
from pygbif import species
key = species.name_backbone(name = 'Ursus americanus', rank='species')['usageKey']
occurrences.search(taxonKey = key)

# Search by dataset key
occurrences.search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', limit=20)

# Search by catalog number
occurrences.search(catalogNumber="49366", limit=20)
```

```
# occurrences.search(catalogNumber=["49366","Bird.27847588"], limit=20)

# Use paging parameters (limit and start) to page. Note the different results
# for the two queries below.
occurrences.search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', offset=10,␣
↪limit=5)
occurrences.search(datasetKey='7b5d6a48-f762-11e1-a439-00145eb45e9a', offset=20,␣
↪limit=5)

# Many dataset keys
# occurrences.search(datasetKey=["50c9509d-22c7-4a22-a47d-8c48425ef4a7",
↪"7b5d6a48-f762-11e1-a439-00145eb45e9a"], limit=20)

# Search by collector name
res = occurrences.search(recordedBy="smith", limit=20)
[ x['recordedBy'] for x in res['results'] ]

# Many collector names
# occurrences.search(recordedBy=["smith","BJ Stacey"], limit=20)

# Search for many species
splist = ['Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa']
keys = [ species.name_suggest(x)[0]['key'] for x in splist ]
out = [ occurrences.search(taxonKey = x, limit=1) for x in keys ]
[ x['results'][0]['speciesKey'] for x in out ]

# Search - q parameter
occurrences.search(q="kingfisher", limit=2)

# Search on latitidue and longitude
occurrences.search(decimalLatitude=50, decimalLongitude=10, limit=2)

# Search on a bounding box
## in well known text format
occurrences.search(geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))
↪', limit=20)
from pygbif import species
key = species.name_suggest(q='Aesculus hippocastanum')[0]['key']
occurrences.search(taxonKey=key, geometry='POLYGON((30.1 10.1, 10 20, 20 40, 40␣
↪40, 30.1 10.1))', limit=20)

# Search on country
occurrences.search(country='US', limit=20)
occurrences.search(country='FR', limit=20)
occurrences.search(country='DE', limit=20)

# Get only occurrences with lat/long data
occurrences.search(taxonKey=key, hasCoordinate=True, limit=20)

# Get only occurrences that were recorded as living specimens
occurrences.search(taxonKey=key, basisOfRecord="LIVING_SPECIMEN",␣
↪hasCoordinate=True, limit=20)

# Get occurrences for a particular eventDate
occurrences.search(taxonKey=key, eventDate="2013", limit=20)
occurrences.search(taxonKey=key, year="2013", limit=20)
occurrences.search(taxonKey=key, month="6", limit=20)
```

```
# Get occurrences based on depth
key = species.name_backbone(name='Salmo salar', kingdom='animals')['usageKey']
occurrences.search(taxonKey=key, depth="5", limit=20)

# Get occurrences based on elevation
key = species.name_backbone(name='Puma concolor', kingdom='animals')['usageKey']
occurrences.search(taxonKey=key, elevation=50, hasCoordinate=True, limit=20)

# Get occurrences based on institutionCode
occurrences.search(institutionCode="TLMF", limit=20)

# Get occurrences based on collectionCode
occurrences.search(collectionCode="Floristic Databases MV - Higher Plants",␣
↪limit=20)

# Get only those occurrences with spatial issues
occurrences.search(taxonKey=key, hasGeospatialIssue=True, limit=20)

# Search using a query string
occurrences.search(q="kingfisher", limit=20)

# Range queries
## See Detail for parameters that support range queries
### this is a range depth, with lower/upper limits in character string
occurrences.search(depth='50,100')

## Range search with year
occurrences.search(year='1999,2000', limit=20)

## Range search with latitude
occurrences.search(decimalLatitude='29.59,29.6')

# Search by specimen type status
## Look for possible values of the typeStatus parameter looking at the typestatus␣
↪dataset
occurrences.search(typeStatus = 'allotype')

# Search by specimen record number
## This is the record number of the person/group that submitted the data, not GBIF
↪'s numbers
## You can see that many different groups have record number 1, so not super␣
↪helpful
occurrences.search(recordNumber = 1)

# Search by last time interpreted: Date the record was last modified in GBIF
## The lastInterpreted parameter accepts ISO 8601 format dates, including
## yyyy, yyyy-MM, yyyy-MM-dd, or MM-dd. Range queries are accepted for␣
↪lastInterpreted
occurrences.search(lastInterpreted = '2014-04-01')

# Search by continent
## One of africa, antarctica, asia, europe, north_america, oceania, or south_
↪america
occurrences.search(continent = 'south_america')
occurrences.search(continent = 'africa')
occurrences.search(continent = 'oceania')
occurrences.search(continent = 'antarctica')
```

```
# Search for occurrences with images
occurrences.search(mediatype = 'StillImage')
occurrences.search(mediatype = 'MovingImage')
x = occurrences.search(mediatype = 'Sound')
[z['media'] for z in x['results']]

# Query based on issues
occurrences.search(taxonKey=1, issue='DEPTH_UNLIKELY')
occurrences.search(taxonKey=1, issue=['DEPTH_UNLIKELY','COORDINATE_ROUNDED'])
# Show all records in the Arizona State Lichen Collection that cant be matched to␣
↪the GBIF
# backbone properly:
occurrences.search(datasetKey='84c0e1a0-f762-11e1-a439-00145eb45e9a', issue=[
↪'TAXON_MATCH_NONE','TAXON_MATCH_HIGHERRANK'])

# If you pass in an invalid polygon you get hopefully informative errors
### the WKT string is fine, but GBIF says bad polygon
wkt = 'POLYGON((-178.59375 64.83258989321493,-165.9375 59.24622380205539,
-147.3046875 59.065977905449806,-130.78125 51.04484764446178,-125.859375 36.
↪70806354647625,
-112.1484375 23.367471303759686,-105.1171875 16.093320185359257,-86.8359375 9.
↪23767076398516,
-82.96875 2.9485268155066175,-82.6171875 -14.812060061226388,-74.8828125 -18.
↪849111862023985,
-77.34375 -47.661687803329166,-84.375 -49.975955187343295,174.7265625 -50.
↪649460483096114,
179.296875 -42.19189902447192,-176.8359375 -35.634976650677295,176.8359375 -31.
↪835565983656227,
163.4765625 -6.528187613695323,152.578125 1.894796132058301,135.703125 4.
↪702353722559447,
127.96875 15.077427674847987,127.96875 23.689804541429606,139.921875 32.
↪06861069132688,
149.4140625 42.65416193033991,159.2578125 48.3160811030533,168.3984375 57.
↪019804336633165,
178.2421875 59.95776046458139,-179.6484375 61.16708631440347,-178.59375 64.
↪83258989321493))'
occurrences.search(geometry = wkt)
```

occurrences.**get**(*key*, ***kwargs*)

Gets details for a single, interpreted occurrence

Parameters **key** – [int] A GBIF occurrence key

Returns A dictionary, of results

Usage:

```
from pygbif import occurrences
occurrences.get(key = 252408386)
```

occurrences.**get_verbatim**(*key*, ***kwargs*)

Gets a verbatim occurrence record without any interpretation

Parameters **key** – [int] A GBIF occurrence key

Returns A dictionary, of results

Usage:

```
from pygbif import occurrences
occurrences.get_verbatim(key = 252408386)
```

occurrences.**get_fragment**(*key*, ***kwargs*)
Get a single occurrence fragment in its raw form (xml or json)

Parameters **key** – [int] A GBIF occurrence key

Returns A dictionary, of results

Usage:

```
from pygbif import occurrences
occurrences.get_fragment(key = 1052909293)
```

occurrences.**count**(*taxonKey=None*, *basisOfRecord=None*, *country=None*, *isGeoreferenced=None*, *datasetKey=None*, *publishingCountry=None*, *typeStatus=None*, *issue=None*, *year=None*, ***kwargs*)
Returns occurrence counts for a predefined set of dimensions

Parameters

- **taxonKey** – [int] A GBIF occurrence identifier
- **basisOfRecord** – [str] A GBIF occurrence identifier
- **country** – [str] A GBIF occurrence identifier
- **isGeoreferenced** – [bool] A GBIF occurrence identifier
- **datasetKey** – [str] A GBIF occurrence identifier
- **publishingCountry** – [str] A GBIF occurrence identifier
- **typeStatus** – [str] A GBIF occurrence identifier
- **issue** – [str] A GBIF occurrence identifier
- **year** – [int] A GBIF occurrence identifier

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count(taxonKey = 3329049)
occurrences.count(country = 'CA')
occurrences.count(isGeoreferenced = True)
occurrences.count(basisOfRecord = 'OBSERVATION')
```

occurrences.**count_basisofrecord**(***kwargs*)
Lists occurrence counts by basis of record.

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count_basisofrecord()
```

occurrences.**count_year**(*year*, ***kwargs*)
Lists occurrence counts by year

Parameters **`year`** – [int] year range, e.g., "1990,2000". Does not support ranges like "asterisk,2010"

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count_year(year = '1990,2000')
```

`occurrences.`**`count_datasets`**(*taxonKey=None*, *country=None*, ***kwargs*)

Lists occurrence counts for datasets that cover a given taxon or country

Parameters

- **`taxonKey`** – [int] Taxon key
- **`country`** – [str] A country, two letter code

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count_datasets(country = "DE")
```

`occurrences.`**`count_countries`**(*publishingCountry*, ***kwargs*)

Lists occurrence counts for all countries covered by the data published by the given country

Parameters **`publishingCountry`** – [str] A two letter country code

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count_countries(publishingCountry = "DE")
```

`occurrences.`**`count_schema`**(***kwargs*)

List the supported metrics by the service

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count_schema()
```

`occurrences.`**`count_publishingcountries`**(*country*, ***kwargs*)

Lists occurrence counts for all countries that publish data about the given country

Parameters **`country`** – [str] A country, two letter code

Returns dict

Usage:

```
from pygbif import occurrences
occurrences.count_publishingcountries(country = "DE")
```

`occurrences.`**`download_meta`**(*key*, ***kwargs*)

Retrieves the occurrence download metadata by its unique key.

Parameters

- **key** – [str] A key generated from a request, like that from *download*
- ****kwargs** – Further named arguments passed on to *requests.get*

Usage:

```
from pygbif import occurrences as occ
occ.download_meta(key = "0003970-140910143529206")
occ.download_meta(key = "0000099-140929101555934")
```

occurrences.**download_list**(*user=None*, *pwd=None*, *limit=20*, *start=0*, ***kwargs*)
Lists the downloads created by a user.

Parameters

- **user** – [str] A user name, look at env var "GBIF_USER" first
- **pwd** – [str] Your password, look at env var "GBIF_PWD" first
- **limit** – [int] Number of records to return. Default: 20
- **start** – [int] Record number to start at. Default: 0
- ****kwargs** – Further named arguments passed on to *requests.get*

Usage:

```
from pygbif import occurrences as occ
occ.download_list(user = "sckott")
occ.download_list(user = "sckott", limit = 5)
occ.download_list(user = "sckott", start = 21)
```

occurrences.**download_get**(*key*, *path='.'*, *overwrite=False*, ***kwargs*)
Get a download from GBIF.

Parameters

- **key** – [str] A key generated from a request, like that from *download*
- **path** – [str] Path to write zip file to. Default: ".", with a *.zip* appended to the end.
- ****kwargs** – Further named arguments passed on to *requests.get*

Downloads the zip file to a directory you specify on your machine. We stream the zip data to a file. This function only downloads the file. See *download_import* to open a downloaded file in Python. The speed of this function is of course proportional to the size of the file to download, and affected by your internet connection speed. For example, a 58 MB file on my machine took about 26 seconds.

Usage:

```
from pygbif import occurrences as occ
occ.download_get("0000066-140928181241064")
occ.download_get("0003983-140910143529206")
```

registry module

registry.**datasets**(*data='all'*, *type=None*, *uuid=None*, *query=None*, *id=None*, *limit=100*, *start=None*, ***kwargs*)
Search for datasets and dataset metadata.

Parameters

- **data** – [str] The type of data to get. Default: 'all'

- **type** – [str] Type of dataset, options include 'OCCURRENCE', etc.
- **uuid** – [str] UUID of the data node provider. This must be specified if data is anything other than 'all'.
- **query** – [str] Query term(s). Only used when *data = 'all'*
- **id** – [int] A metadata document id.

References http://www.gbif.org/developer/registry#datasets

Usage:

```
from pygbif import registry
registry.datasets(limit=5)
registry.datasets(type="OCCURRENCE")
registry.datasets(uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
registry.datasets(data='contact', uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
registry.datasets(data='metadata', uuid="a6998220-7e3a-485d-9cd6-73076bd85657")
registry.datasets(data='metadata', uuid="a6998220-7e3a-485d-9cd6-73076bd85657",␣
↪id=598)
registry.datasets(data=['deleted','duplicate'])
registry.datasets(data=['deleted','duplicate'], limit=1)
```

registry.**dataset_metrics**(*uuid*, ***kwargs*)

Get details on a GBIF dataset.

Parameters uuid – [str] One or more dataset UUIDs. See examples.

References: http://www.gbif.org/developer/registry#datasetMetrics

Usage:

```
from pygbif import registry
registry.dataset_metrics(uuid='3f8a1297-3259-4700-91fc-acc4170b27ce')
registry.dataset_metrics(uuid='66dd0960-2d7d-46ee-a491-87b9adcfe7b1')
registry.dataset_metrics(uuid=['3f8a1297-3259-4700-91fc-acc4170b27ce', '66dd0960-
↪2d7d-46ee-a491-87b9adcfe7b1'])
```

registry.**installations**(*data='all'*, *uuid=None*, *query=None*, *identifier=None*, *identifierType=None*, *limit=100*, *start=None*, ***kwargs*)

Installations metadata.

Parameters

- **data** – [str] The type of data to get. Default is all data. If not 'all', then one or more of 'contact', 'endpoint', 'dataset', 'comment', 'deleted', 'nonPublishing'.
- **uuid** – [str] UUID of the data node provider. This must be specified if data is anything other than 'all'.
- **query** – [str] Query nodes. Only used when *data='all'*. Ignored otherwise.

References: http://www.gbif.org/developer/registry#installations

Usage:

```
from pygbif import registry
registry.installations(limit=5)
registry.installations(query="france")
registry.installations(uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4")
registry.installations(data='contact', uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4
↪")
```

```
registry.installations(data='contact', uuid="2e029a0c-87af-42e6-87d7-f38a50b78201
↪")
registry.installations(data='endpoint', uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4
↪")
registry.installations(data='dataset', uuid="b77901f9-d9b0-47fa-94e0-dd96450aa2b4
↪")
registry.installations(data='deleted')
registry.installations(data='deleted', limit=2)
registry.installations(data=['deleted','nonPublishing'], limit=2)
registry.installations(identifierType='DOI', limit=2)
```

`registry.`**`networks`**(*data='all'*, *uuid=None*, *query=None*, *identifier=None*, *identifierType=None*, *limit=100*, *start=None*, ***kwargs*)

Networks metadata.

Parameters

- **`data`** – [str] The type of data to get. Default: 'all'
- **`uuid`** – [str] UUID of the data network provider. This must be specified if data is anything other than 'all'.
- **`query`** – [str] Query networks. Only used when *data* = *'all'*. Ignored otherwise.

References: http://www.gbif.org/developer/registry#networks

Returns A dict

Usage:

```
from pygbif import registry
registry.networks(limit=5)
registry.networks(uuid='16ab5405-6c94-4189-ac71-16ca3b753df7')
registry.networks(data='endpoint', uuid='16ab5405-6c94-4189-ac71-16ca3b753df7')
```

`registry.`**`nodes`**(*data='all'*, *uuid=None*, *query=None*, *identifier=None*, *identifierType=None*, *limit=100*, *start=None*, *isocode=None*, ***kwargs*)

Nodes metadata.

Parameters

- **`data`** – [str] The type of data to get. Default: 'all'
- **`uuid`** – [str] UUID of the data node provider. This must be specified if data is anything other than 'all'.
- **`query`** – [str] Query nodes. Only used when *data* = *'all'*
- **`isocode`** – [str] A 2 letter country code. Only used if *data* = *'country'*.

References http://www.gbif.org/developer/registry#nodes

Usage:

```
from pygbif import registry
registry.nodes(limit=5)
registry.nodes(uuid="1193638d-32d1-43f0-a855-8727c94299d8")
registry.nodes(data='identifier', uuid="03e816b3-8f58-49ae-bc12-4e18b358d6d9")
registry.nodes(data=['identifier','organization','comment'], uuid="03e816b3-8f58-
↪49ae-bc12-4e18b358d6d9")

uuids = ["8cb55387-7802-40e8-86d6-d357a583c596","02c40d2a-1cba-4633-90b7-
↪e36e5e97aba8",
"7a17efec-0a6a-424c-b743-f715852c3c1f","b797ce0f-47e6-4231-b048-6b62ca3b0f55",
"1193638d-32d1-43f0-a855-8727c94299d8","d3499f89-5bc0-4454-8cdb-60bead228a6d",
"cdc9736d-5ff7-4ece-9959-3c744360cdb3","a8b16421-d80b-4ef3-8f22-098b01a89255",
"8df8d012-8e64-4c8a-886e-521a3bdfa623","b35cf8f1-748d-467a-adca-4f9170f20a4e",
"03e816b3-8f58-49ae-bc12-4e18b358d6d9","073d1223-70b1-4433-bb21-dd70afe3053b",
"07dfe2f9-5116-4922-9a8a-3e0912276a72","086f5148-c0a8-469b-84cc-cce5342f9242",
"0909d601-bda2-42df-9e63-a6d51847ebce","0e0181bf-9c78-4676-bdc3-54765e661bb8",
"109aea14-c252-4a85-96e2-f5f4d5d088f4","169eb292-376b-4cc6-8e31-9c2c432de0ad",
"1e789bc9-79fc-4e60-a49e-89dfc45a7188","1f94b3ca-9345-4d65-afe2-4bace93aa0fe"]

[ registry.nodes(data='identifier', uuid=x) for x in uuids ]
```

species module

species.**name_backbone**(*name*, *rank=None*, *kingdom=None*, *phylum=None*, *clazz=None*, *order=None*, *family=None*, *genus=None*, *strict=False*, *verbose=False*, *start=None*, *limit=100*, ***kwargs*)

Lookup names in the GBIF backbone taxonomy.

Parameters

- **name** – [str] Full scientific name potentially with authorship (required)
- **rank** – [str] The rank given as our rank enum. (optional)
- **kingdom** – [str] If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
- **phylum** – [str] If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
- **class** – [str] If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
- **order** – [str] If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
- **family** – [str] If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
- **genus** – [str] If provided default matching will also try to match against this if no direct match is found for the name alone. (optional)
- **strict** – [bool] If True it (fuzzy) matches only the given name, but never a taxon in the upper classification (optional)
- **verbose** – [bool] If True show alternative matches considered which had been rejected.

A list for a single taxon with many slots (with verbose=False - default), or a list of length two, first element for the suggested taxon match, and a data.frame with alternative name suggestions resulting from fuzzy matching (with verbose=True).

If you don't get a match GBIF gives back a list of length 3 with slots synonym, confidence, and matchType='NONE'.

reference: http://www.gbif.org/developer/species#searching

Usage:

```
from pygbif import species
species.name_backbone(name='Helianthus annuus', kingdom='plants')
species.name_backbone(name='Helianthus', rank='genus', kingdom='plants')
species.name_backbone(name='Poa', rank='genus', family='Poaceae')

# Verbose - gives back alternatives
species.name_backbone(name='Helianthus annuus', kingdom='plants', verbose=True)

# Strictness
species.name_backbone(name='Poa', kingdom='plants', verbose=True, strict=False)
species.name_backbone(name='Helianthus annuus', kingdom='plants', verbose=True,␣
↪strict=True)

# Non-existent name
species.name_backbone(name='Aso')

# Multiple equal matches
species.name_backbone(name='Oenante')
```

species.**name_suggest**(*q=None*, *datasetKey=None*, *rank=None*, *fields=None*, *start=None*, *limit=100*, ***kwargs*)

A quick and simple autocomplete service that returns up to 20 name usages by doing prefix matching against the scientific name. Results are ordered by relevance.

References: http://www.gbif.org/developer/species#searching

Parameters

- **q** – [str] Simple search parameter. The value for this parameter can be a simple word or a phrase. Wildcards can be added to the simple word parameters only, e.g. q=*puma* (Required)
- **datasetKey** – [str] Filters by the checklist dataset key (a uuid, see examples)
- **rank** – [str] A taxonomic rank. One of class, cultivar, cultivar_group, domain, family, form, genus, informal, infrageneric_name, infraorder, infraspecific_name, infrasubspecific_name, kingdom, order, phylum, section, series, species, strain, subclass, subfamily, subform, subgenus, subkingdom, suborder, subphylum, subsection, subseries, subspecies, subtribe, subvariety, superclass, superfamily, superorder, superphylum, suprageneric_name, tribe, unranked, or variety.

Returns A dictionary, of results

Usage:

```
from pygbif import species
species.name_suggest(q='Puma concolor')
x = species.name_suggest(q='Puma')
x['data']
x['hierarchy']
species.name_suggest(q='Puma', rank="genus")
species.name_suggest(q='Puma', rank="subspecies")
species.name_suggest(q='Puma', rank="species")
species.name_suggest(q='Puma', rank="infraspecific_name")
species.name_suggest(q='Puma', limit=2)
```

species.**name_parser**(*name*, ***kwargs*)

Parse taxon names using the GBIF name parser

Parameters **name** – [str] A character vector of scientific names. (required) reference: http://www.gbif.org/developer/species#parser

Usage:

```
from pygbif import species
species.name_parser('x Agropogon littoralis')
species.name_parser(['Arrhenatherum elatius var. elatius',
  'Secale cereale subsp. cereale', 'Secale cereale ssp. cereale',
  'Vanessa atalanta (Linnaeus, 1758)'])
```

Changelog

0.1.4 (2016-06-04)

- Added variable types throughout docs
- Changed default *limit* value to 300 for *occurrences.search* method
- *tox* now included, via @xrotwang (#20)
- Added more registry methods (#11)
- Started occurrence download methods (#16
- Added more names methods (#18)
- All requests now send user-agent headers with *requests* and *pygbif* versions (#13)
- Bug fix for *occurrences.download_get* (#23)
- Fixed bad example for *occurrences.get* (#22)
- Fixed wheel to be universal for 2 and 3 (#10)
- Improved documentation a lot, autodoc methods now

0.1.1 (2015-11-03)

- Fixed distribution for pypi

0.1.0 (2015-11-02)

- First release

License

MIT

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index