pygbif Documentation

Release 0.2.2.1

Scott Chamberlain

Jan 24, 2018

Installation

Python client for the GBIF API

Source on GitHub at sckott/pygbif

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 pygbif in our issue tracker.

CHAPTER 2

Installation

2.1 Installation guide

2.1.1 Installing pygbif

Stable from pypi

```
pip install pygbif
```

Development version

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

Installation guide How to install pygbif.

CHAPTER 3

Docs

3.1 Frequently Asked Questions

3.1.1 What other GBIF clients are out there?

- R: rgbif

3.2 Usecases

3.2.1 Use case 1: Get occurrence data for a set of taxonomic names

Load library

```
from pygbif import species as species
from pygbif import occurrences as occ
```

First, get GBIF backbone taxonomic keys

```
splist = ['Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa',
  'Ursus americanus', 'Pinus conorta', 'Poa annuus']
keys = [ species.name_backbone(x)['usageKey'] for x in splist ]
```

Then, get a count of occurrence records for each taxon, and pull out number of records found for each taxon

```
out = [ occ.search(taxonKey = x, limit=0)['count'] for x in keys ]
```

Make a dict of species names and number of records, sorting in descending order

```
x = dict(zip(splist, out))
sorted(x.items(), key=lambda z:z[1], reverse=True)
```

Frequently Asked Questions Frequently asked questions.

Usecases Usecases for pygbif.

CHAPTER 4

Modules

4.1 pygbif modules

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

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

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

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

4.2 occurrence module

occurrence module API:

- *search*
- *get*
- *get_verbatim*
- *get_fragment*
- *count*
- *count_basisofrecord*
- *count_year*
- *count_datasets*
- *count_countries*
- *count_schema*

- *count_publishingcountries*
- *download*
- *download_meta*
- *download_list*
- *download_get*

Example usage:

```
from pygbif import occurrences as occ
occ.search(taxonKey = 3329049)
occ.get(key = 252408386)
occ.count(isGeoreferenced = True)
occ.download('basisOfRecord = LITERATURE')
occ.download('taxonKey = 3119195')
occ.download('decimalLatitude > 50')
occ.download_list(user = "sckott", limit = 5)
occ.download_meta(key = "0000099-140929101555934")
occ.download_get("0000066-140928181241064")
```

4.2.1 occurrences API

occurrences.**search**(*taxonKey=None*, *repatriated=None*, *kingdomKey=None*, *phylumKey=None*, *classKey=None*, *orderKey=None*, *familyKey=None*, *genusKey=None*, *subgenusKey=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*, *spellCheck=None*, *mediatype=None*, *limit=300*, *offset=0*, *establishmentMeans=None*, *facet=None*, *facetMincount=None*, *facetMultiselect=None*, ***kwargs*)

Search GBIF occurrences

Parameters

- **taxonKey** – [int] A GBIF occurrence identifier
- **q** – [str] Simple search parameter. The value for this parameter can be a simple word or a phrase.
- **spellCheck** – [bool] If `True` ask GBIF to check your spelling of the value passed to the `search` parameter. IMPORTANT: This only checks the input to the `search` parameter, and no others. Default: `False`
- **repatriated** – [str] Searches for records whose publishing country is different to the country where the record was recorded in
- **kingdomKey** – [int] Kingdom classification key
- **phylumKey** – [int] Phylum classification key
- **classKey** – [int] Class classification key
- **orderKey** – [int] Order classification key
- **familyKey** – [int] Family classification key

- **genusKey** – [int] Genus classification key
- **subgenusKey** – [int] Subgenus classification key
- **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 POLYGON, or MULTIPOLYGON. 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. `hasGeospatialIssue=TRUE` returns only those records with spatial issues while `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** – [bool] 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 longitude. 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`
- **facet** – [str] a character vector of length 1 or greater
- **establishmentMeans** – [str] EstablishmentMeans, possible values include: INTRODUCED, INVASIVE, MANAGED, NATIVE, NATURALISED, UNCERTAIN
- **facetMincount** – [int] minimum number of records to be included in the faceting results
- **facetMultiselect** – [bool] Set to `true` to still return counts for values that are not currently filtered. See examples. Default: `false`

Returns A dictionary

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 `scientificName="..."`)
# you are getting synonyms too. The results for using `scientifcName` and
# `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 offset) 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=20)
## spell check - only works with the `search` parameter
### spelled correctly - same result as above call
occurrences.search(q = "kingfisher", limit=20, spellCheck = True)
### spelled incorrectly - stops with suggested spelling
occurrences.search(q = "kajsdkla", limit=20, spellCheck = True)
### spelled incorrectly - stops with many suggested spellings
###    and number of results for each
```

```
occurrences.search(q = "helir", limit=20, spellCheck = True)

# 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)
## multipolygon
wkt = 'MULTIPOLYGON(((-123 38, -123 43, -116 43, -116 38, -123 38)),((-97 41, -97␣
↪45, -93 45, -93 41, -97 41)))'
occurrences.search(geometry = wkt, 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)

# Faceting
## return no occurrence records with limit=0
x = occurrences.search(facet = "country", limit = 0)
x['facets']

## also return occurrence records
x = occurrences.search(facet = "establishmentMeans", limit = 10)
x['facets']
x['results']

## multiple facet variables
x = occurrences.search(facet = ["country", "basisOfRecord"], limit = 10)
x['results']
x['facets']
x['facets']['country']
x['facets']['basisOfRecord']
x['facets']['basisOfRecord']['count']

## set a minimum facet count
x = occurrences.search(facet = "country", facetMincount = 30000000L, limit = 0)
x['facets']

## paging per each faceted variable
### do so by passing in variables like "country" + "_facetLimit" = "country_
↪facetLimit"
### or "country" + "_facetOffset" = "country_facetOffset"
x = occurrences.search(
  facet = ["country", "basisOfRecord", "hasCoordinate"],
  country_facetLimit = 3,
  basisOfRecord_facetLimit = 6,
  limit = 0
)
x['facets']

# requests package options
## There's an acceptable set of requests options (['timeout', 'cookies', 'auth',
## 'allow_redirects', 'proxies', 'verify', 'stream', 'cert']) you can pass
## in via **kwargs, e.g., set a timeout
x = occurrences.search(timeout = 1)
```

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 = 1258202889)
occurrences.get(key = 1227768771)
occurrences.get(key = 1227769518)
```

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 = 1258202889)
occurrences.get_verbatim(key = 1227768771)
occurrences.get_verbatim(key = 1227769518)
```

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.get_fragment(key = 1227768771)
occurrences.get_fragment(key = 1227769518)
```

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**(*queries*, *user=None*, *pwd=None*, *email=None*, *pred_type='and'*)

Spin up a download request for GBIF occurrence data.

Parameters

- **queries** (*str or list*) – One or more of query arguments to kick of a download job. See Details.
- **pred_type** – (character) One of `equals` (=), `and` (&), *or*‘ (|), `lessThan` (<), `lessThanOrEquals` (<=), `greaterThan` (>), `greaterThanOrEquals` (>=), `in`, `within`, `not` (!), `like`
- **user** – (character) User name within GBIF’s website. Required. Set in your env vars with the option `GBIF_USER`
- **pwd** – (character) User password within GBIF’s website. Required. Set in your env vars with the option `GBIF_PWD`
- **email** – (character) Email address to recieve download notice done email. Required. Set in your env vars with the option `GBIF_EMAIL`

Argument passed have to be passed as character (e.g., `country = US`), with a space between key (`country`), operator (=), and value (`US`). See the `type` parameter for possible options for the operator. This character string is parsed internally.

Acceptable arguments to `...` (args) are:

- taxonKey = `TAXON_KEY`
- scientificName = `SCIENTIFIC_NAME`
- country = `COUNTRY`
- publishingCountry = `PUBLISHING_COUNTRY`
- hasCoordinate = `HAS_COORDINATE`
- hasGeospatialIssue = `HAS_GEOSPATIAL_ISSUE`
- typeStatus = `TYPE_STATUS`
- recordNumber = `RECORD_NUMBER`
- lastInterpreted = `LAST_INTERPRETED`
- continent = `CONTINENT`
- geometry = `GEOMETRY`

- basisOfRecord = BASIS_OF_RECORD
- datasetKey = DATASET_KEY
- eventDate = EVENT_DATE
- catalogNumber = CATALOG_NUMBER
- year = YEAR
- month = MONTH
- decimalLatitude = DECIMAL_LATITUDE
- decimalLongitude = DECIMAL_LONGITUDE
- elevation = ELEVATION
- depth = DEPTH
- institutionCode = INSTITUTION_CODE
- collectionCode = COLLECTION_CODE
- issue = ISSUE
- mediatype = MEDIA_TYPE
- recordedBy = RECORDED_BY
- repatriated = REPATRIATED

See the API docs http://www.gbif.org/developer/occurrence#download for more info, and the predicates docs http://www.gbif.org/developer/occurrence#predicates

GBIF has a limit of 12,000 characters for download queries - so if you're download request is really, really long and complex, consider breaking it up into multiple requests by one factor or another.

Returns A dictionary, of results

Usage:

```
from pygbif import occurrences as occ

occ.download('basisOfRecord = LITERATURE')
occ.download('taxonKey = 3119195')
occ.download('decimalLatitude > 50')
occ.download('elevation >= 9000')
occ.download('decimalLatitude >= 65')
occ.download('country = US')
occ.download('institutionCode = TLMF')
occ.download('catalogNumber = Bird.27847588')

res = occ.download(['taxonKey = 7264332', 'hasCoordinate = TRUE'])

# pass output to download_meta for more information
occ.download_meta(occ.download('decimalLatitude > 75'))

# Multiple queries
gg = occ.download(['decimalLatitude >= 65',
                   'decimalLatitude <= -65'], type='or')
gg = occ.download(['depth = 80', 'taxonKey = 2343454'],
                   type='or')
```

```
# Repratriated data for Costa Rica
occ.download(['country = CR', 'repatriated = true'])
```

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

Retrieves the occurrence download metadata by its unique key. Further named arguments passed on to `requests.get` can be included as additional arguments

Parameters **key** – [str] A key generated from a request, like that from `download`

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*, *offset=0*)

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`
- **offset** – [int] Record number to start at. Default: `0`

Usage:

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

occurrences.**download_get**(*key*, *path='.'*, ***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. The speed of this function is of course proportional to the size of the file to download, and affected by your internet connection speed.

This function only downloads the file. To open and read it, see https://github.com/BelgianBiodiversityPlatform/python-dwca-reader

Usage:

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

4.3 registry module

registry module API:

- *organizations*
- *nodes*
- *networks*
- *installations*
- *datasets*
- *dataset_metrics*
- *dataset_suggest*
- *dataset_search*

Example usage:

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

4.3.1 registry API

registry.**datasets**(*data='all'*, *type=None*, *uuid=None*, *query=None*, *id=None*, *limit=100*, *offset=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.**dataset_suggest**(*q=None*, *type=None*, *keyword=None*, *owningOrg=None*, *publishingOrg=None*, *hostingOrg=None*, *publishingCountry=None*, *decade=None*, *limit=100*, *offset=None*, ***kwargs*)

Search that returns up to 20 matching datasets. Results are ordered by relevance.

Parameters

- **q** – [str] Query term(s) for full text search. 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*`
- **type** – [str] Type of dataset, options include OCCURRENCE, etc.
- **keyword** – [str] Keyword to search by. Datasets can be tagged by keywords, which you can search on. The search is done on the merged collection of tags, the dataset keywordCollections and temporalCoverages. SEEMS TO NOT BE WORKING ANYMORE AS OF 2016-09-02.
- **owningOrg** – [str] Owning organization. A uuid string. See *organizations()*
- **publishingOrg** – [str] Publishing organization. A uuid string. See *organizations()*
- **hostingOrg** – [str] Hosting organization. A uuid string. See *organizations()*
- **publishingCountry** – [str] Publishing country.
- **decade** – [str] Decade, e.g., 1980. Filters datasets by their temporal coverage broken down to decades. Decades are given as a full year, e.g. 1880, 1960, 2000, etc, and will return datasets wholly contained in the decade as well as those that cover the entire decade or more. Facet by decade to get the break down, e.g. `/search?facet=DECADE&facet_only=true` (see example below)
- **limit** – [int] Number of results to return. Default: `300`
- **offset** – [int] Record to start at. Default: `0`

Returns A dictionary

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

Usage:

```
from pygbif import registry
registry.dataset_suggest(q="Amazon", type="OCCURRENCE")

# Suggest datasets tagged with keyword "france".
registry.dataset_suggest(keyword="france")

# Suggest datasets owned by the organization with key
# "07f617d0-c688-11d8-bf62-b8a03c50a862" (UK NBN).
registry.dataset_suggest(owningOrg="07f617d0-c688-11d8-bf62-b8a03c50a862")
```

```
# Fulltext search for all datasets having the word "amsterdam" somewhere in
# its metadata (title, description, etc).
registry.dataset_suggest(q="amsterdam")

# Limited search
registry.dataset_suggest(type="OCCURRENCE", limit=2)
registry.dataset_suggest(type="OCCURRENCE", limit=2, offset=10)

# Return just descriptions
registry.dataset_suggest(type="OCCURRENCE", limit = 5, description=True)

# Search by decade
registry.dataset_suggest(decade=1980, limit = 30)
```

registry.**dataset_search**(*q=None*, *type=None*, *keyword=None*, *owningOrg=None*, *publishingOrg=None*, *hostingOrg=None*, *decade=None*, *publishingCountry=None*, *facet=None*, *facetMincount=None*, *facetMultiselect=None*, *hl=False*, *limit=100*, *offset=None*, ***kwargs*)

Full text search across all datasets. Results are ordered by relevance.

Parameters

- **q** – [str] Query term(s) for full text search. 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*`
- **type** – [str] Type of dataset, options include OCCURRENCE, etc.
- **keyword** – [str] Keyword to search by. Datasets can be tagged by keywords, which you can search on. The search is done on the merged collection of tags, the dataset keywordCollections and temporalCoverages. SEEMS TO NOT BE WORKING ANYMORE AS OF 2016-09-02.
- **owningOrg** – [str] Owning organization. A uuid string. See *organizations()*
- **publishingOrg** – [str] Publishing organization. A uuid string. See *organizations()*
- **hostingOrg** – [str] Hosting organization. A uuid string. See *organizations()*
- **publishingCountry** – [str] Publishing country.
- **decade** – [str] Decade, e.g., 1980. Filters datasets by their temporal coverage broken down to decades. Decades are given as a full year, e.g. 1880, 1960, 2000, etc, and will return datasets wholly contained in the decade as well as those that cover the entire decade or more. Facet by decade to get the break down, e.g. `/search?facet=DECADE&facet_only=true` (see example below)
- **facet** – [str] A list of facet names used to retrieve the 100 most frequent values for a field. Allowed facets are: type, keyword, publishingOrg, hostingOrg, decade, and publishingCountry. Additionally subtype and country are legal values but not yet implemented, so data will not yet be returned for them.
- **facetMincount** – [str] Used in combination with the facet parameter. Set facetMincount={#} to exclude facets with a count less than {#}, e.g. http://api.gbif.org/v1/dataset/search?facet=type&limit=0&facetMincount=10000 only shows the type value 'OCCURRENCE' because 'CHECKLIST' and 'METADATA' have counts less than 10000.
- **facetMultiselect** – [bool] Used in combination with the facet parameter. Set facetMultiselect=True to still return counts for values that are not currently filtered, e.g. http://api.gbif.org/v1/dataset/search?facet=type&limit=0&type=CHECKLIST&

facetMultiselect=true still shows type values 'OCCURRENCE' and 'METADATA' even though type is being filtered by type=CHECKLIST

- **hl** – [bool] Set `hl=True` to highlight terms matching the query when in fulltext search fields. The highlight will be an emphasis tag of class 'gbifH1' e.g. http://api.gbif.org/v1/dataset/search?q=plant&hl=true Fulltext search fields include: title, keyword, country, publishing country, publishing organization title, hosting organization title, and description. One additional full text field is searched which includes information from metadata documents, but the text of this field is not returned in the response.
- **limit** – [int] Number of results to return. Default: `300`
- **offset** – [int] Record to start at. Default: `0`

Note Note that you can pass in additional faceting parameters on a per field basis. For example, if you want to limit the numbef of facets returned from a field `foo` to 3 results, pass in `foo_facetLimit = 3`. GBIF does not allow all per field parameters, but does allow some. See also examples.

Returns A dictionary

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

Usage:

```
from pygbif import registry
# Gets all datasets of type "OCCURRENCE".
registry.dataset_search(type="OCCURRENCE", limit = 10)

# Fulltext search for all datasets having the word "amsterdam" somewhere in
# its metadata (title, description, etc).
registry.dataset_search(q="amsterdam", limit = 10)

# Limited search
registry.dataset_search(type="OCCURRENCE", limit=2)
registry.dataset_search(type="OCCURRENCE", limit=2, offset=10)

# Search by decade
registry.dataset_search(decade=1980, limit = 10)

# Faceting
## just facets
registry.dataset_search(facet="decade", facetMincount=10, limit=0)

## data and facets
registry.dataset_search(facet="decade", facetMincount=10, limit=2)

## many facet variables
registry.dataset_search(facet=["decade", "type"], facetMincount=10, limit=0)

## facet vars
### per variable paging
x = registry.dataset_search(
        facet = ["decade", "type"],
        decade_facetLimit = 3,
        type_facetLimit = 3,
        limit = 0
)

## highlight
```

```
x = registry.dataset_search(q="plant", hl=True, limit = 10)
[ z['description'] for z in x['results'] ]
```

registry.**installations**(*data='all'*, *uuid=None*, *q=None*, *identifier=None*, *identifierType=None*, *limit=100*, *offset=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`.
- **q** – [str] Query nodes. Only used when `data='all'`. Ignored otherwise.
- **identifier** – [fixnum] The value for this parameter can be a simple string or integer, e.g. identifier=120
- **identifierType** – [str] Used in combination with the identifier parameter to filter identifiers by identifier type: `DOI`, `FTP`, `GBIF_NODE`, `GBIF_PARTICIPANT`, `GBIF_PORTAL`, `HANDLER`, `LSID`, `UNKNOWN`, `URI`, `URL`, `UUID`
- **limit** – [int] Number of results to return. Default: `100`
- **offset** – [int] Record to start at. Default: `0`

Returns A dictionary

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

Usage:

```
from pygbif import registry
registry.installations(limit=5)
registry.installations(q="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*, *q=None*, *identifier=None*, *identifierType=None*, *limit=100*, *offset=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`.
- **q** – [str] Query networks. Only used when `data = 'all'`. Ignored otherwise.

- **identifier** – [fixnum] The value for this parameter can be a simple string or integer, e.g. identifier=120
- **identifierType** – [str] Used in combination with the identifier parameter to filter identifiers by identifier type: `DOI`, `FTP`, `GBIF_NODE`, `GBIF_PARTICIPANT`, `GBIF_PORTAL`, `HANDLER`, `LSID`, `UNKNOWN`, `URI`, `URL`, `UUID`
- **limit** – [int] Number of results to return. Default: `100`
- **offset** – [int] Record to start at. Default: `0`

Returns A dictionary

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

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*, *q=None*, *identifier=None*, *identifierType=None*, *limit=100*, *offset=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`.
- **q** – [str] Query nodes. Only used when `data = 'all'`
- **identifier** – [fixnum] The value for this parameter can be a simple string or integer, e.g. identifier=120
- **identifierType** – [str] Used in combination with the identifier parameter to filter identifiers by identifier type: `DOI`, `FTP`, `GBIF_NODE`, `GBIF_PARTICIPANT`, `GBIF_PORTAL`, `HANDLER`, `LSID`, `UNKNOWN`, `URI`, `URL`, `UUID`
- **limit** – [int] Number of results to return. Default: `100`
- **offset** – [int] Record to start at. Default: `0`
- **isocode** – [str] A 2 letter country code. Only used if `data = 'country'`.

Returns A dictionary

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

Usage:

```
from pygbif import registry
registry.nodes(limit=5)
registry.nodes(identifier=120)
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 ]
```

registry.**organizations**(*data='all'*, *uuid=None*, *q=None*, *identifier=None*, *identifierType=None*, *limit=100*, *offset=None*, ***kwargs*)

organizations metadata.

Parameters

- **data** – [str] The type of data to get. Default is all data. If not `all`, then one or more of `contact`, `endpoint`, `identifier`, `tag`, `machineTag`, `comment`, `hostedDataset`, `ownedDataset`, `deleted`, `pending`, `nonPublishing`.
- **uuid** – [str] UUID of the data node provider. This must be specified if data is anything other than `all`.
- **q** – [str] Query nodes. Only used when `data='all'`. Ignored otherwise.
- **identifier** – [fixnum] The value for this parameter can be a simple string or integer, e.g. identifier=120
- **identifierType** – [str] Used in combination with the identifier parameter to filter identifiers by identifier type: `DOI`, `FTP`, `GBIF_NODE`, `GBIF_PARTICIPANT`, `GBIF_PORTAL`, `HANDLER`, `LSID`, `UNKNOWN`, `URI`, `URL`, `UUID`
- **limit** – [int] Number of results to return. Default: `100`
- **offset** – [int] Record to start at. Default: `0`

Returns A dictionary

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

Usage:

```
from pygbif import registry
registry.organizations(limit=5)
registry.organizations(q="france")
registry.organizations(identifier=120)
registry.organizations(uuid="e2e717bf-551a-4917-bdc9-4fa0f342c530")
registry.organizations(data='contact', uuid="e2e717bf-551a-4917-bdc9-4fa0f342c530
↪")
registry.organizations(data='endpoint', uuid="e2e717bf-551a-4917-bdc9-4fa0f342c530
↪")
registry.organizations(data='deleted')
registry.organizations(data='deleted', limit=2)
registry.organizations(data=['deleted','nonPublishing'], limit=2)
registry.organizations(identifierType='DOI', limit=2)
```

4.4 species module

species module API:

- *name_backbone*
- *name_suggest*
- *name_usage*
- *name_lookup*
- *name_parser*

Example usage:

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

4.4.1 species API

species.**name_backbone**(*name*, *rank=None*, *kingdom=None*, *phylum=None*, *clazz=None*, *order=None*, *family=None*, *genus=None*, *strict=False*, *verbose=False*, *offset=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.
- **offset** – [int] Record to start at. Default: `0`
- **limit** – [int] Number of results to return. Default: `100`

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*, *limit=100*, *offset=None*, ***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.

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`.
- **`limit`** – [fixnum] Number of records to return. Maximum: `1000`. (optional)
- **`offset`** – [fixnum] Record number to start at. (optional)

Returns A dictionary

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

Usage:

```
from pygbif import species

species.name_suggest(q='Puma concolor')
x = species.name_suggest(q='Puma')
```

```
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_lookup**(*q=None*, *rank=None*, *higherTaxonKey=None*, *status=None*, *isExtinct=None*, *habitat=None*, *nameType=None*, *datasetKey=None*, *nomenclaturalStatus=None*, *limit=100*, *offset=None*, *facet=False*, *facetMincount=None*, *facetMultiselect=None*, *type=None*, *hl=False*, *verbose=False*, ***kwargs*)

Lookup names in all taxonomies in GBIF.

This service uses fuzzy lookup so that you can put in partial names and you should get back those things that match. See examples below.

Parameters

- **q** – [str] Query term(s) for full text search (optional)
- **rank** – [str] 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, VARIETY (optional)
- **verbose** – [bool] If True show alternative matches considered which had been rejected.
- **higherTaxonKey** – [str] Filters by any of the higher Linnean rank keys. Note this is within the respective checklist and not searching nub keys across all checklists (optional)
- **status** – [str] (optional) Filters by the taxonomic status as one of:
  - ACCEPTED
  - DETERMINATION_SYNONYM Used for unknown child taxa referred to via spec, ssp, ...
  - DOUBTFUL Treated as accepted, but doubtful whether this is correct.
  - HETEROTYPIC_SYNONYM More specific subclass of SYNONYM.
  - HOMOTYPIC_SYNONYM More specific subclass of SYNONYM.
  - INTERMEDIATE_RANK_SYNONYM Used in nub only.
  - MISAPPLIED More specific subclass of SYNONYM.
  - PROPARTE_SYNONYM More specific subclass of SYNONYM.
  - SYNONYM A general synonym, the exact type is unknown.
- **isExtinct** – [bool] Filters by extinction status (e.g. isExtinct=True)
- **habitat** – [str] Filters by habitat. One of: marine, freshwater, or terrestrial (optional)
- **nameType** – [str] (optional) Filters by the name type as one of:
  - BLACKLISTED surely not a scientific name.
  - CANDIDATUS Candidatus is a component of the taxonomic name for a bacterium that cannot be maintained in a Bacteriology Culture Collection.

  - CULTIVAR a cultivated plant name.
  - DOUBTFUL doubtful whether this is a scientific name at all.
  - HYBRID a hybrid formula (not a hybrid name).
  - INFORMAL a scientific name with some informal addition like "cf." or indetermined like Abies spec.
  - SCINAME a scientific name which is not well formed.
  - VIRUS a virus name.
  - WELLFORMED a well formed scientific name according to present nomenclatural rules.
- **datasetKey** – [str] Filters by the dataset's key (a uuid) (optional)
- **nomenclaturalStatus** – [str] Not yet implemented, but will eventually allow for filtering by a nomenclatural status enum
- **limit** – [fixnum] Number of records to return. Maximum: 1000. (optional)
- **offset** – [fixnum] Record number to start at. (optional)
- **facet** – [str] A list of facet names used to retrieve the 100 most frequent values for a field. Allowed facets are: datasetKey, higherTaxonKey, rank, status, isExtinct, habitat, and nameType. Additionally threat and nomenclaturalStatus are legal values but not yet implemented, so data will not yet be returned for them. (optional)
- **facetMincount** – [str] Used in combination with the facet parameter. Set facetMincount={#} to exclude facets with a count less than {#}, e.g. http://bit.ly/1bMdByP only shows the type value ACCEPTED because the other statuses have counts less than 7,000,000 (optional)
- **facetMultiselect** – [bool] Used in combination with the facet parameter. Set facetMultiselect=True to still return counts for values that are not currently filtered, e.g. http://bit.ly/19YLXPO still shows all status values even though status is being filtered by status=ACCEPTED (optional)
- **type** – [str] Type of name. One of occurrence, checklist, or metadata. (optional)
- **hl** – [bool] Set hl=True to highlight terms matching the query when in fulltext search fields. The highlight will be an emphasis tag of class gbifH1 e.g. q='plant', hl=True. Fulltext search fields include: title, keyword, country, publishing country, publishing organization title, hosting organization title, and description. One additional full text field is searched which includes information from metadata documents, but the text of this field is not returned in the response. (optional)

Returns A dictionary

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

Usage:

```
from pygbif import species

# Look up names like mammalia
species.name_lookup(q='mammalia')

# Paging
species.name_lookup(q='mammalia', limit=1)
species.name_lookup(q='mammalia', limit=1, offset=2)
```

```
# large requests, use offset parameter
first = species.name_lookup(q='mammalia', limit=1000)
second = species.name_lookup(q='mammalia', limit=1000, offset=1000)

# Get all data and parse it, removing descriptions which can be quite long
species.name_lookup('Helianthus annuus', rank="species", verbose=True)

# Get all data and parse it, removing descriptions field which can be quite long
out = species.name_lookup('Helianthus annuus', rank="species")
res = out['results']
[ z.pop('descriptions', None) for z in res ]
res

# Fuzzy searching
species.name_lookup(q='Heli', rank="genus")

# Limit records to certain number
species.name_lookup('Helianthus annuus', rank="species", limit=2)

# Query by habitat
species.name_lookup(habitat = "terrestrial", limit=2)
species.name_lookup(habitat = "marine", limit=2)
species.name_lookup(habitat = "freshwater", limit=2)

# Using faceting
species.name_lookup(facet='status', limit=0, facetMincount='70000')
species.name_lookup(facet=['status', 'higherTaxonKey'], limit=0, facetMincount=
↪'700000')

species.name_lookup(facet='nameType', limit=0)
species.name_lookup(facet='habitat', limit=0)
species.name_lookup(facet='datasetKey', limit=0)
species.name_lookup(facet='rank', limit=0)
species.name_lookup(facet='isExtinct', limit=0)

# text highlighting
species.name_lookup(q='plant', hl=True, limit=30)

# Lookup by datasetKey
species.name_lookup(datasetKey='3f8a1297-3259-4700-91fc-acc4170b27ce')
```

species.**name_usage**(*key=None*, *name=None*, *data='all'*, *language=None*, *datasetKey=None*, *uuid=None*, *sourceId=None*, *rank=None*, *shortname=None*, *limit=100*, *offset=None*, ***kwargs*)

Lookup details for specific names in all taxonomies in GBIF.

Parameters

- **key** – [fixnum] A GBIF key for a taxon
- **name** – [str] Filters by a case insensitive, canonical namestring, e.g. 'Puma concolor'
- **data** – [str] The type of data to get. Default: `all`. Options: `all`, `verbatim`, `name`, `parents`, `children`, `related`, `synonyms`, `descriptions`, `distributions`, `media`, `references`, `speciesProfiles`, `vernacularNames`, `typeSpecimens`, `root`
- **language** – [str] Language, default is english

- **datasetKey** – [str] Filters by the dataset's key (a uuid)
- **uuid** – [str] A uuid for a dataset. Should give exact same results as datasetKey.
- **sourceId** – [fixnum] Filters by the source identifier.
- **rank** – [str] Taxonomic rank. Filters by taxonomic rank as 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, VARIETY
- **shortname** – [str] A short name..need more info on this?
- **limit** – [fixnum] Number of records to return. Default: 100. Maximum: 1000. (optional)
- **offset** – [fixnum] Record number to start at. (optional)

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

Usage:

```
from pygbif import species

species.name_usage(key=1)

# Name usage for a taxonomic name
species.name_usage(name='Puma', rank="GENUS")

# All name usages
species.name_usage()

# References for a name usage
species.name_usage(key=2435099, data='references')

# Species profiles, descriptions
species.name_usage(key=3119195, data='speciesProfiles')
species.name_usage(key=3119195, data='descriptions')
species.name_usage(key=2435099, data='children')

# Vernacular names for a name usage
species.name_usage(key=3119195, data='vernacularNames')

# Limit number of results returned
species.name_usage(key=3119195, data='vernacularNames', limit=3)

# Search for names by dataset with datasetKey parameter
species.name_usage(datasetKey="d7dddbf4-2cf0-4f39-9b2a-bb099caae36c")

# Search for a particular language
species.name_usage(key=3119195, language="FRENCH", data='vernacularNames')
```

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)'])
```

pygbif modules Introduction to pygbif modules.

occurrence module The occurrence module: core GBIF occurrence data, including count, search, and download APIs.

registry module The registry module: including datasets, installations, networks, nodes, and organizations.

species module The species module: including name search, lookup, suggest, usage, and backbone search.

CHAPTER 5

All the rest

5.1 Changelog

5.1.1 0.2.0 (2016-10-18)

- Download methods much improved (#16) (#27) thanks @jlegind @stijnvanhoey @peterdesmet !
- MULTIPOLYGON now supported in *geometry* parameter (#35)
- Fixed docs for *occurrences.get*, and *occurrences.get_verbatim*, *occurrences.get_fragment* and demo that used occurrence keys that no longer exist in GBIF (#39)
- Added *organizations* method to *registry* module (#12)
- Added remainder of datasets methods: *registry.dataset_search* (including faceting support (#37)) and *registry.dataset_suggest*, for the */dataset/search* and */dataset/suggest* routes, respectively (#40)
- Added remainder of species methods: *species.name_lookup* (including faceting support (#38)) and *species.name_usage*, for the */species/search* and */species* routes, respectively (#18)
- Added more tests to cover new methods
- Changed *species.name_suggest* to give back data stucture as received from GBIF. We used to parse out the classification data, but for simplicity and speed, that is left up to the user now.
- *start* parameter in *species.name_suggest*, *occurrences.download_list*, *registry.organizations*, *registry.nodes*, *registry.networks*, and *registry.installations*, changed to *offset* to match GBIF API and match usage throughout remainder of *pygbif*

5.1.2 0.1.5.4 (2016-10-01)

- Added many new *occurrence.search* parameters, including *repatriated*, *kingdomKey*, *phylumKey*, *classKey*, *orderKey*, *familyKey*, *genusKey*, *subgenusKey*, *establishmentMeans*, *facet*, *facetMincount*, *facetMultiselect*, and support for facet paging via ***kwargs* (#30) (#34)

- Fixes to **kwargs in *occurrence.search* so that facet parameters can be parsed correctly and *requests* GET request options are collected correctly (#36)
- Added *spellCheck* parameter to *occurrence.search* that goes along with the *q* parameter to optionally spell check full text searches (#31)

5.1.3 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

5.1.4 0.1.1 (2015-11-03)

- Fixed distribution for pypi

5.1.5 0.1.0 (2015-11-02)

- First release

5.2 Contributors

- Scott Chamberlain
- Robert Forkel
- Jan Legind
- Stijn Van Hoey
- Peter Desmet

5.3 Contributing

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

5.3.1 Bug reports

Please report bug reports on our issue tracker.

5.3.2 Feature requests

Please put feature requests on our issue tracker.

5.3.3 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

5.3.4 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
```

5.4 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/

5.5 LICENSE

Changelog See what has changed in recent pygbif versions.

Contributors pygbif contributors.

Contributing Learn how to contribute to the pygbif project.

Contributor Code of Conduct Expected behavior in this community. By participating in this project you agree to abide by its terms.

LICENSE The pygbif license.

5.6 Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index