MyGene.py Documentation

Release v3.1.0

Chunlei Wu

Dec 18, 2018

Contents

1 Requirements 3

2 Optional dependencies 5

3 Installation 7

4 Version history 9

5 Tutorial 11

6 API 13

7 Indices and tables 19

Python Module Index 21

MyGene.Info provides simple-to-use REST web services to query/retrieve gene annotation data. It's designed with simplicity and performance emphasized. *mygene*, is an easy-to-use Python wrapper to access MyGene.Info services.

Note: As of v3.1.0, mygene Python package is now a thin wrapper of underlying biothings_client package, a universal Python client for all BioThings APIs, including MyGene.info. The installation of mygene will install biothings_client automatically. The following code snippets are essentially equivalent:

- Continue using mygene package

```
In [1]: import mygene
In [2]: mg = mygene.MyGeneInfo()
```

- Use biothings_client package directly

```
In [1]: from biothings_client import get_client
In [2]: mg = get_client('gene')
```

After that, the use of `mg` instance is exactly the same.

CHAPTER 1

Requirements

Python >=2.7 (including python3)

(Python 2.6 might still work, not it’s not supported any more since v3.1.0.)

biothings_client (>=0.2.0, install using “pip install biothings_client”)

CHAPTER 2

Optional dependencies

pandas (install using “pip install pandas”) is required for returning a list of gene objects as DataFrame.

CHAPTER 3

Installation

Option 1 pip install mygene

Option 2 download/extract the source code and run:

```
python setup.py install
```

Option 3 install the latest code directly from the repository:

```
pip install -e git+https://github.com/biothings/mygene.py#egg=mygene
```

CHAPTER 4

Version history

CHANGES.txt

CHAPTER 5

Tutorial

- ID mapping using mygene module in Python

CHAPTER 6

API

`mygene.alwayslist(value)`

If input value if not a list/tuple type, return it as a single value list.

Example:

```
>>> x = 'abc'
>>> for xx in alwayslist(x):
...     print xx
>>> x = ['abc', 'def']
>>> for xx in alwayslist(x):
...     print xx
```

class `mygene.MyGeneInfo(url=None)`

`clear_cache()`

Clear the globally installed cache.

`findgenes(id_li, **kwargs)`

Deprecated since version 2.0.0.

Use `querymany()` instead. It's kept here as an alias of `querymany()` method.

`get_fields(search_term=None, verbose=True)`

Return all available fields can be return from MyGene.info services.

This is a wrapper for http://mygene.info/metadata/fields

Parameters **`search_term`** – an optional string to search (case insensitive) for matching field names. If not provided, all available fields will be returned.

Example:

```
>>> mv.get_fields()
>>> mv.get_fields("uniprot")
>>> mv.get_fields("refseq")
>>> mv.get_fields("kegg")
```

Hint: This is useful to find out the field names you need to pass to **fields** parameter of other methods.

`getgene` (*_id*, *fields=None*, ***kwargs*)

Return the gene object for the give geneid. This is a wrapper for GET query of "/gene/<geneid>" service.

Parameters

- **`geneid`** – entrez/ensembl gene id, entrez gene id can be either a string or integer
- **`fields`** – fields to return, a list or a comma-separated string. If **fields="all"**, all available fields are returned
- **`species`** – optionally, you can pass comma-separated species names or taxonomy ids
- **`email`** – optionally, pass your email to help us to track usage
- **`filter`** – alias for **fields** parameter

Returns a gene object as a dictionary, or None if geneid is not valid.

Ref http://docs.mygene.info/en/latest/doc/data.html for available fields, extra *kwargs* and more.

Example:

```
>>> mg.getgene(1017, email='abc@example.com')
>>> mg.getgene('1017', fields='symbol,name,entrezgene,refseq')
>>> mg.getgene('1017', fields='symbol,name,entrezgene,refseq.rna')
>>> mg.getgene('1017', fields=['symbol', 'name', 'pathway.kegg'])
>>> mg.getgene('ENSG00000123374', fields='all')
```

Hint: The supported field names passed to **fields** parameter can be found from any full gene object (when **fields="all"**). Note that field name supports dot notation for nested data structure as well, e.g. you can pass "refseq.rna" or "pathway.kegg".

`getgenes` (*ids*, *fields=None*, ***kwargs*)

Return the list of gene objects for the given list of geneids. This is a wrapper for POST query of "/gene" service.

Parameters

- **`geneids`** – a list/tuple/iterable or comma-separated entrez/ensembl gene ids
- **`fields`** – fields to return, a list or a comma-separated string. If **fields="all"**, all available fields are returned
- **`species`** – optionally, you can pass comma-separated species names or taxonomy ids
- **`email`** – optionally, pass your email to help us to track usage
- **`filter`** – alias for fields
- **`as_dataframe`** – if True, return object as DataFrame (requires Pandas).
- **`df_index`** – if True (default), index returned DataFrame by 'query', otherwise, index by number. Only applicable if as_dataframe=True.

Returns a list of gene objects or a pandas DataFrame object (when **as_dataframe** is True)

Ref http://mygene.info/doc/annotation_service.html for available fields, extra *kwargs* and more.

Example:

```
>>> mg.getgenes([1017, '1018','ENSG00000148795'], email='abc@example.com')
>>> mg.getgenes([1017, '1018','ENSG00000148795'], fields="entrezgene,uniprot")
>>> mg.getgenes([1017, '1018','ENSG00000148795'], fields="all")
>>> mg.getgenes([1017, '1018','ENSG00000148795'], as_dataframe=True)
```

Hint: A large list of more than 1000 input ids will be sent to the backend web service in batches (1000 at a time), and then the results will be concatenated together. So, from the user-end, it's exactly the same as passing a shorter list. You don't need to worry about saturating our backend servers.

metadata(*verbose=True*, ***kwargs*)

Return a dictionary of MyGene.info metadata.

Example:

```
>>> metadata = mg.metadata
```

query(*q*, ***kwargs*)

Return the query result. This is a wrapper for GET query of "/query?q=<query>" service.

Parameters

- **q** – a query string, detailed query syntax here
- **fields** – fields to return, a list or a comma-separated string. If **fields="all"**, all available fields are returned
- **species** – optionally, you can pass comma-separated species names or taxonomy ids. Default: human,mouse,rat.
- **size** – the maximum number of results to return (with a cap of 1000 at the moment). Default: 10.
- **skip** – the number of results to skip. Default: 0.
- **sort** – Prefix with "-" for descending order, otherwise in ascending order. Default: sort by matching scores in decending order.
- **entrezonly** – if True, return only matching entrez genes, otherwise, including matching Ensemble-only genes (those have no matching entrez genes).
- **email** – optionally, pass your email to help us to track usage
- **as_dataframe** – if True, return object as DataFrame (requires Pandas).
- **df_index** – if True (default), index returned DataFrame by 'query', otherwise, index by number. Only applicable if as_dataframe=True.
- **fetch_all** – if True, return a generator to all query results (unsorted). This can provide a very fast return of all hits from a large query. Server requests are done in blocks of 1000 and yielded individually. Each 1000 block of results must be yielded within 1 minute, otherwise the request will expire on the server side.

Returns a dictionary with returned gene hits or a pandas DataFrame object (when **as_dataframe** is True)

Ref http://mygene.info/doc/query_service.html for available fields, extra *kwargs* and more.

Example:

```
>>> mg.query('cdk2')
>>> mg.query('reporter:1000_at')
>>> mg.query('symbol:cdk2', species='human')
>>> mg.query('symbol:cdk*', species=10090, size=5, as_dataframe=True)
>>> mg.query('q=chrX:151073054-151383976', species=9606)
```

querymany (*qterms*, *scopes=None*, ***kwargs*)

Return the batch query result. This is a wrapper for POST query of "/query" service.

Parameters

- **qterms** – a list/tuple/iterable of query terms, or a string of comma-separated query terms.
- **scopes** – type of types of identifiers, either a list or a comma-separated fields to specify type of input qterms, e.g. "entrezgene", "entrezgene,symbol", ["ensemblgene", "symbol"]. Refer to official MyGene.info docs for full list of fields.
- **fields** – fields to return, a list or a comma-separated string. If **fields="all"**, all available fields are returned
- **species** – optionally, you can pass comma-separated species names or taxonomy ids. Default: human,mouse,rat.
- **entrezonly** – if True, return only matching entrez genes, otherwise, including matching Ensemble-only genes (those have no matching entrez genes).
- **returnall** – if True, return a dict of all related data, including dup. and missing qterms
- **verbose** – if True (default), print out infomation about dup and missing qterms
- **email** – optionally, pass your email to help us to track usage
- **as_dataframe** – if True, return object as DataFrame (requires Pandas).
- **df_index** – if True (default), index returned DataFrame by 'query', otherwise, index by number. Only applicable if as_dataframe=True.

Returns a list of gene objects or a pandas DataFrame object (when **as_dataframe** is True)

Ref http://mygene.info/doc/query_service.html for available fields, extra *kwargs* and more.

Example:

```
>>> mg.querymany(['DDX26B', 'CCDC83'], scopes='symbol', species=9606)
>>> mg.querymany(['1255_g_at', '1294_at', '1316_at', '1320_at'], scopes=
↪'reporter')
>>> mg.querymany(['NM_003466', 'CDK2', 695, '1320_at', 'Q08345'],
...              scopes='refseq,symbol,entrezgene,reporter,uniprot', species=
↪'human')
>>> mg.querymany(['1255_g_at', '1294_at', '1316_at', '1320_at'], scopes=
↪'reporter',
...              fields='ensembl.gene,symbol', as_dataframe=True)
```

Hint: *querymany()* is perfect for doing id mappings.

Hint: Just like *getgenes()*, passing a large list of ids (>1000) to *querymany()* is perfectly fine.

set_caching (*cache_db=None*, *verbose=True*, ***kwargs*)

Installs a local cache for all requests.

cache_db is the path to the local sqlite cache database.

`stop_caching``()`
Stop caching.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

mygene, 13

Index

A

alwayslist() (in module mygene), 13

C

clear_cache() (mygene.MyGeneInfo method), 13

F

findgenes() (mygene.MyGeneInfo method), 13

G

get_fields() (mygene.MyGeneInfo method), 13
getgene() (mygene.MyGeneInfo method), 14
getgenes() (mygene.MyGeneInfo method), 14

M

metadata() (mygene.MyGeneInfo method), 15
mygene (module), 13
MyGeneInfo (class in mygene), 13

Q

query() (mygene.MyGeneInfo method), 15
querymany() (mygene.MyGeneInfo method), 16

S

set_caching() (mygene.MyGeneInfo method), 16
stop_caching() (mygene.MyGeneInfo method), 17