PyHGNC Documentation

Release 0.2.4

Christian Ebeling
Andrej Konotopez

Mar 11, 2022

Contents

for version: 0.2.4

PyHGNC is a python software interface developed by the Department of Bioinformatics at the Fraunhofer Institute for Algorithms and Scientific Computing SCAI for the data provided by the European Bioinformatics Institute (EMBL-EBI) on their HGNC website. Thanks to the significant and important work of the HUGO Gene Nomenclature Committee the scientific community has an essential standardised nomenclature for human genes in hand. Because in many software projects a local installation with fast programmatic access is required, we have developed a Python library which allows to access and query HGNC data locally with very limited programming skills.

For the impatient user: on your console

```
pip install pyhgnc
pyhgnc update
```

... in your Python console

```
import pyhgnc
query = pyhgnc.query()
query.hgnc()
```

The content of HGNC made easy accessible by PyHGNC supports successfully scientists in the IMI funded projects AETIONOMY and PHAGO. It is used for the identification of potential drugs in complex disease networks with several thousand relationships compiled from BEL statements.

Aim of this software project is to provide an programmatic access to locally stored HGNC data and allow a filtered export in several formats used in the scientific community. Query functions allow to search in the data and use it as *pandas.DataFrame* in Jupyter notebooks. We will focus our software development on the analysis and extension of biological disease knowledge networks. PyHGNC is an ongoing project and needs improvement. We are happy if you want to support our project or start a scientific cooperation with us.

ToDo: Add Figure of PyHGNC ER

Fig. 1: ER model of PyHGNC database

- supported by IMI, AETIONOMY, PHAGO.

CHAPTER 1

Installation

1.1 System requirements

After complete installation of HGNC (gene symbols and names) and HCOP (orthology) data by *PyHGNC* ~1,441,250 rows in 22 tables need only ~380 MB of disk storage (depending on the used RDMS).

Tests were performed on *Ubuntu 16.04, 4 x Intel Core i7-6560U CPU @ 2.20Ghz* with *16 GiB of RAM*. In general PyHGNC should work also on other systems like Windows, other Linux distributions or Mac OS. Installation were complete after ~4 min. For systems with low memory the option *–low_memory* was added in the update method.

1.2 Supported databases

PyHGNC uses SQLAlchemy to cover a wide spectrum of RDMSs (Relational database management systems). For best performance MySQL or MariaDB is recommended. But if you have no possibility to install software on your system, SQLite - which needs no further installation - also works. The following RDMSs are supported (by SQLAlchemy):

1. Firebird
2. Microsoft SQL Server
3. MySQL / MariaDB
4. Oracle
5. PostgreSQL
6. SQLite
7. Sybase

1.3 Install software

The following instructions are written for Linux/MacOS. The way you install python software on Windows could be different.

Often it makes sense to avoid conflicts with other python installations by using different virtual environments. Read here about easy setup and management of different virtual environments.

- If you want to install *pyhgnc* system wide use superuser (sudo for Ubuntu):

```
sudo pip install pyhgnc
```

- If you have no sudo rights install as user

```
pip install --user pyhgnc
```

- If you want to make sure you install *pyhgnc* in python3 environment:

```
sudo python3 -m pip install pyhgnc
```

1.3.1 MySQL/MariaDB setup

In general you don't have to setup any database, because *pyhgnc* uses file based SQLite by default. But we strongly recommend to use MySQL/MariaDB.

Log in MySQL/MariaDB as root user and create a new database, create a user, assign the rights and flush privileges.

```
CREATE DATABASE pyhgnc CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON pyhgnc.* TO 'pyhgnc_user'@'%' IDENTIFIED BY 'pyhgnc_passwd';
FLUSH PRIVILEGES;
```

The simplest way to set the configurations of *pyhgnc* for MySQL/MariaDB is to use the command . . .

```
pyhgnc mysql
```

. . . and accept all default values.

Another way is to open a python shell and set the MySQL configuration. If you have not changed anything in the SQL statements . . .

```
import pyhgnc
pyhgnc.set_mysql_connection()
```

If you have used you own settings, please adapt the following command to you requirements.

```
import pyhgnc
pyhgnc.set_mysql_connection(host='localhost', user='pyhgnc_user', passwd='pyhgnc_
↪passwd', db='pyhgnc')
```

1.3.2 Updating

During the updating process PyHGNC will download HGNC and HCOP files from the EBI ftp server.

Downloaded files will take no space on your disk after the update process.

To update from command line or terminal:

```
pyhgnc update
```

Update options are available aswell, type *pyhgnc update –help* to get a full list with descriptions.

To update from Python shell:

```
import pyhgnc
pyhgnc.update()
```

1.4 Changing database configuration

Following functions allow to change the connection to your RDBMS (relational database management system). The connection settings will be used by default on the next time `pyhgnc` is executed.

To set a new MySQL/MariaDB connection use the interactive command line interface (bash, terminal, cmd) . . .

```
pyhgnc mysql
```

. . . or in Python shell . . .

```
import pyhgnc
pyhgnc.set_mysql_connection(host='localhost', user='pyhgnc_user', passwd='pyhgnc_
↪passwd', db='pyhgnc')
```

To set connection to other database systems use the *database.set_connection()*.

For more information about connection strings go to the SQLAlchemy documentation.

Examples for valid connection strings are:

- mysql+pymysql://user:passwd@localhost/database?charset=utf8
- postgresql://scott:tiger@localhost/mydatabase
- mssql+pyodbc://user:passwd@database
- oracle://user:passwd@127.0.0.1:1521/database
- Linux: sqlite:////absolute/path/to/database.db
- Windows: sqlite:///C:\path\to\database.db

You could use the following code to connect *pyhgnc* to an oracle database:

```
import pyhgnc
pyhgnc.set_connection('oracle://user:passwd@127.0.0.1:1521/database')
```

CHAPTER 2

Tutorial

In this tutorial we will use a new python environment using *virualenvwrapper* and make a fresh install of *pyhgnc*. Then we will perform an update and store the data in a SQLite database. Afterwards we will perform some queries to analyze the data we received and stored in our database.

2.1 Setup

1. New environment
2. install pyhgnc

2.2 Update

1. Setup connection
2. perform update

2.3 Query

Use python shell to perform some basic queries.

CHAPTER 3

Query functions

Contents

- *ccds*
- *pubmed*
- *ena*
- *enzyme*
- *lsdb*

3.1 Before you query

3.1.1 1. You can use % as a wildcard.

```
import pyhgnc
query = pyhgnc.query()

# exact search
query.hgnc(name='amyloid beta precursor protein')

# starts with 'amyloid beta'
query.hgnc(name='amyloid beta %')

# ends with 'precursor protein'
query.hgnc(name='% precursor protein')

# contains 'precursor'
query.hgnc(name='%precursor%')
```

3.1.2 2. *limit* to restrict number of results

```
import pyhgnc
query = pyhgnc.query()

query.hgnc(limit=10)
```

Use an offset by paring a tuple *(page_number, number_of_results_per_page)* to the parameter *limit*.

page_number starts with 0!

```
import pyhgnc
query = pyhgnc.query()

# first page with 3 results (every page have 3 results)
query.hgnc(limit=(0,3))
# fourth page with 10 results (every page have 10 results)
query.hgnc(limit=(4,10))
```

3.1.3 3. Return `pandas.DataFrame` as result

This is very useful if you want to profit from amazing pandas functions.

```
import pyhgnc
query = pyhgnc.query()

query.hgnc(as_df=True)
```

3.1.4 4. show all columns as dict

```
import pyhgnc
query = pyhgnc.query()

first_entry = query.hgnc(limit=1)[0]
first_entry.to_dict()
```

3.1.5 5. Return single values with key name

```
import pyhgnc
query = pyhgnc.query()

query.hgnc(name='%kinase')[0].name
```

3.1.6 6. Access to the linked data models (1-n, n-m)

From results of *pyhgnc.query().hgnc()* you can access

- alias_symbols
- alias_names
- rgds
- omims
- ccdss
- lsdbs
- orthology_predictions
- enzymes
- gene_families
- refseq_accessions
- mgds
- uniprots
- pubmeds
- enas

```
import pyhgnc
query = pyhgnc.query()

r = query.hgnc(limit=1)[0]
```

(continues on next page)

(continued from previous page)

```
r.alias_symbols
r.alias_names
r.rgds
r.omims
r.ccdss
r.lsdbs
r.orthology_predictions
r.enzymes
r.gene_families
r.refseq_accessions
r.mgds
r.uniprots
r.pubmeds
r.enas
```

But for example from *pyhgnc.query().uniprot()* you can go back to hgnc

```
import pyhgnc
query = pyhgnc.query()

uniprot = query.uniprot(uniprotid='Q9BTE6')[0]
uniprot.hgncs
# [AARSD1, PTGES3L-AARSD1]
# following is crazy but possible, again go back to ec_number
uniprot.hgncs[0].uniprots
# [Q9BTE6]
```

3.1.7 7. HGNC identifier and symbol is available in all methods

Hint: In all query functions (except *hgnc*) you have the parameters - hgnc_identifier - hgnc_symbol even it is not part of the model.

```
import pyhgnc
query = pyhgnc.query()

query.alias_symbol(hgnc_identifier=620)
# [AD1]
query.alias_symbol(hgnc_symbol='APP')
# [AD1]
```

3.2 hgnc

```
import pyhgnc
query = pyhgnc.query()

query.hgnc(entrez=503538)
```

Check documentation of *pyhgnc.manager.query.QueryManager.hgnc()* for all available parameters.

3.3 orthology_prediction

```
import pyhgnc
query = pyhgnc.query()

query.orthology_prediction(ortholog_species=10090, hgnc_symbol='APP')
# [10090: amyloid beta (A4) precursor protein: App]
```

Check documentation of *pyhgnc.manager.query.QueryManager.orthology_prediction()* for all available parameters.

3.4 alias_symbol

```
import pyhgnc
query = pyhgnc.query()

result = query.alias_symbol(alias_symbol='AD1')[0]
result.hgnc
# APP
```

Check documentation of *pyhgnc.manager.query.QueryManager.alias_symbol()* for all available parameters.

3.5 alias_name

```
import pyhgnc
query = pyhgnc.query()

result = query.alias_name(alias_name='peptidase nexin-II')[0]
result.hgnc.name
# 'amyloid beta precursor protein'
```

Check documentation of *pyhgnc.manager.query.QueryManager.alias_name()* for all available parameters.

3.6 gene_family

```
import pyhgnc
query = pyhgnc.query()

    result = query.gene_family(family_name='Parkinson%')[0]
result
# 'Parkinson disease associated genes'
result.hgncs
# [ATP13A2, EIF4G1, FBXO7, HTRA2, LRRK2, PARK3, PARK7, PARK10, PARK11, PARK12, PARK16,
↪ PINK1,\
# PLA2G6, PRKN, SNCA, UCHL1, VPS35]
```

Check documentation of *pyhgnc.manager.query.QueryManager.gene_family()* for all available parameters.

3.7 ref_seq

```
import pyhgnc
query = pyhgnc.query()

query.ref_seq(hgnc_symbol='APP')
# [NM_000484]
```

Check documentation of *pyhgnc.manager.query.QueryManager.ref_seq()* for all available parameters.

3.8 rgd

```
import pyhgnc
query = pyhgnc.query()

query.rgd(rgdid=2139)[0].hgncs
# [APP]
```

Check documentation of *pyhgnc.manager.query.QueryManager.rgd()* for all available parameters.

3.9 omim

```
import pyhgnc
query = pyhgnc.query()

query.omim(omimid=104760)[0].hgnc.name
# 'amyloid beta precursor protein'
```

Check documentation of *pyhgnc.manager.query.QueryManager.omim()* for all available parameters.

3.10 mgd

```
import pyhgnc
query = pyhgnc.query()

query.mgd(mgdid=88059)[0].hgncs
# [APP]
```

Check documentation of *pyhgnc.manager.query.QueryManager.mgd()* for all available parameters.

3.11 uniprot

```
import pyhgnc
query = pyhgnc.query()

query.uniprot(uniprotid='P05067')[0].hgncs
# [APP]
```

Check documentation of *pyhgnc.manager.query.QueryManager.uniprot()* for all available parameters.

3.12 ccds

```
import pyhgnc
query = pyhgnc.query()

query.ccds(ccdsid='CCDS13576')[0].hgnc
# APP
```

Check documentation of *pyhgnc.manager.query.QueryManager.ccds()* for all available parameters.

3.13 pubmed

```
import pyhgnc
query = pyhgnc.query()

query.pubmed(hgnc_symbol='A1CF')
# [11815617, 11072063]
```

Check documentation of *pyhgnc.manager.query.QueryManager.pubmed()* for all available parameters.

3.14 ena

```
import pyhgnc
query = pyhgnc.query()

query.ena(hgnc_identifier=620)
# [AD1]
```

Check documentation of *pyhgnc.manager.query.QueryManager.ena()* for all available parameters.

3.15 enzyme

```
import pyhgnc
query = pyhgnc.query()

query.enzyme(hgnc_symbol='PRKCA')
# [2.7.11.1]
```

Check documentation of *pyhgnc.manager.query.QueryManager.enzyme()* for all available parameters.

3.16 lsdb

```
import pyhgnc
query = pyhgnc.query()

query.lsdb(hgnc_symbol='APP')
# [Alzheimer Disease & Frontotemporal Dementia Mutation Database]
```

Check documentation of *pyhgnc.manager.query.QueryManager.lsdb()* for all available parameters.

CHAPTER 4

RESTful API

PyUniProt provides also a RESTful API web server.

Start the server with

```
pyhgnc web
```

Open PyHGNC Web API in a web browser.

CHAPTER 5

HGNC

We want to pay tribute to the HGNC database of human gene names and HUGO Gene Nomenclature Committee team for their amazing resource their provide to the scientific community. `pyhgnc` only provides methods to download and locally query open accessible HGNC data available on EBI ftp server.

5.1 About

Citation from EBI HGNC website [23/11/2017]: "HGNC is the only worldwide authority assigning standardised human gene symbols and names. Its key goals are to provide unique standardised nomenclature for every human gene; to ensure this information is freely available, widely disseminated and universally used; and to coordinate the expansion and utilisation of this nomenclature across vertebrates."

5.2 Citation

Latest HGNC publication:

Gray KA, Yates B, Seal RL, Wright MW, Bruford EA. genenames.org: the HGNC resources in 2015. Nucleic Acids Res. 2015 Jan;43(Database issue):D1079-85. doi: 10.1093/nar/gku1071. PMID:25361968

HGNC Database, HUGO Gene Nomenclature Committee (HGNC), EMBL Outstation - Hinxton, European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridgeshire, CB10 1SD, UK www.genenames.org.

5.3 Links

Link to data: EBI ftp server

Check the HGNC database of human gene names for more information.

CHAPTER 6

HCOP

PyHGNC integrates also HGNC Comparison of Orthology Predictions (HCOP).

6.1 About

HCOP is a tool that integrates orthology assertions predicted for a specified human gene, or set of human genes, by

- eggNOG
- Ensembl Compara
- HGNC
- HomoloGene
- Inparanoid
- NCBI Gene Orthology
- OMA
- OrthoDB
- OrthoMCL
- Panther
- PhylomeDB
- TreeFam
- ZFIN

  An indication of the reliability of a prediction is provided by the number of databases which concur. HCOP was originally designed to show orthology predictions between human and mouse, but has been expanded to include data from chimp, macaque, rat, dog, horse, cow, pig, opossum, platypus, chicken, anole lizard, xenopus, zebrafish, C. elegans, Drosophila and S. cerevisiae, meaning that there are currently 18 genomes available for comparison in HCOP.

6.2 Citation

- **Wright MW, Eyre TA, Lush MJ, Povey S and Bruford EA.** HCOP: The HGNC Comparison of Orthology Predictions Search Tool. Mamm Genome. 2005 Nov; 16(11):827-828. PMID:16284797 PDF
- **Eyre TA, Wright MW, Lush MJ and Bruford EA.** HCOP: a searchable database of human orthology predictions. Brief Bioinform. 2007 Jan;8(1):2-5. PMID: 16951416
- **Gray KA, Yates, B, Seal RL, Wright MW, Bruford EA.** Genenames.org: the HGNC resources in 2015. Nucleic Acids Res. 2015 Jan;43(Database issue):D1079-85. PMID: 25361968

6.3 Links

- HCOP: Orthology Predictions Search

CHAPTER 7

Benchmarks

All benchmarks created on a standard notebook:

- OS: Linux Ubuntu 16.04.2 LTS (xenial)
- Python: 3.5.2
- Hardware: x86_64, Intel(R) Core(TM) i7-6560U CPU @ 2.20GHz, 4 CPUs, Mem 16Gb
- MariaDB: Server version: 10.0.29-MariaDB-0ubuntu0.16.04.1 Ubuntu 16.04

7.1 MySQL/MariaDB

Database created with following command in MySQL/MariaDB as root:

```
CREATE DATABASE pyhgnc CHARACTER SET utf8 COLLATE utf8_general_ci;
```

User created with following command in MySQL/MariaDB:

```
GRANT ALL PRIVILEGES ON pyhgnc.* TO 'pyhgnc_user'@'%' IDENTIFIED BY 'pyhgnc_passwd';
FLUSH PRIVILEGES;
```

7.2 Update

To import HGNC data executed following commands in python console:

```
import pyhgnc
pyhgnc.set_mysql_connection()
pyhgnc.update()
```

The other possibility is to use the command line interface

```
pyhgnc mysql # accept all default values
pyhgnc update
```

- **CPU times:**
  - real 3m52.311s
  - user 3m4.964s
  - sys 0m6.340s

7.3 Low memory option

If you have low memory available please use the –low_memory option in command line

```
pyhgnc mysql # accept all default values
pyhgnc update --low_memory
# or short
pyhgnc update -l
```

- **CPU times:**
  - real 6m40.913s
  - user 5m22.724s
  - sys 0m9.016s

CHAPTER 8

Query

Contents

8.1 Query interface

PyHGNC provides a powerfull query interface for the stored data. It can be accessed from python shell:

```
import pyhgnc
query = pyhgnc.query()
```

You can use the query interface instance to issue a query to any model defined in `pyhgnc.manager.models`:

```
# Issue query on hgnc table:
query.hgnc()

# Issue query on pubmed table:
query.pubmed()
```

Hint: See *Query functions* for more examples and check out *pyhgnc.manager.query.QueryManager* (below) for all possible parameters for the different models.

8.2 Query Manager Reference

class pyhgnc.manager.query.**QueryManager**(*connection=None*, *echo=False*)
Query interface to database.

alias_name(*alias_name=None*, *is_previous_name=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.AliasName* objects in database

Parameters

- **alias_name** (*str or tuple(str) or None*) – alias name(s)
- **is_previous_name** (*bool or tuple(bool) or None*) – flag for 'is previous'
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.AliasSymbol*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.AliasSymbol*) or pandas.DataFrame

alias_symbol(*alias_symbol=None*, *is_previous_symbol=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.AliasSymbol* objects in database

Parameters

- **alias_symbol** (*str or tuple(str) or None*) – alias symbol(s)
- **is_previous_symbol** (*bool or tuple(bool) or None*) – flag for 'is previous'
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.AliasSymbol*)

- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.AliasSymbol*) or pandas.DataFrame

ccds(*ccdsid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.CCDS* objects in database

Parameters

- **ccdsid** (*str or tuple(str) or None*) – Consensus CDS ID(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.CCDS*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.CCDS*) or pandas.DataFrame

ena(*enaid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.ENA* objects in database

Parameters

- **enaid** (*str or tuple(str) or None*) – European Nucleotide Archive (ENA) identifier(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.ENA*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.ENA*) or pandas.DataFrame

enzyme(*ec_number=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.Enzyme* objects in database

Parameters

- **ec_number** (*str or tuple(str) or None*) – Enzyme Commission number (EC number)(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.Enzyme*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.Enzyme*) or pandas.DataFrame

gene_family(*family_identifier=None*, *family_name=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)

Method to query *models.GeneFamily* objects in database

Parameters

- **family_identifier** (*int or tuple(int) or None*) – gene family identifier(s)
- **family_name** (*str or tuple(str) or None*) – gene family name(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.AliasSymbol*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.AliasSymbol*) or pandas.DataFrame

get_model_queries(*query_obj*, *model_queries_config*)

use this if your are searching for a field in the same model

hgnc(*name=None*, *symbol=None*, *identifier=None*, *status=None*, *uuid=None*, *locus_group=None*, *orphanet=None*, *locus_type=None*, *date_name_changed=None*, *date_modified=None*, *date_symbol_changed=None*, *pubmedid=None*, *date_approved_reserved=None*, *ensembl_gene=None*, *horde=None*, *vega=None*, *lncrnadb=None*, *uniprotid=None*, *entrez=None*, *mirbase=None*, *iuphar=None*, *ucsc=None*, *snornabase=None*, *gene_family_name=None*, *mgdid=None*, *pseudogeneorg=None*, *bioparadigmsslc=None*, *locationsortable=None*, *ec_number=None*, *refseq_accession=None*, *merops=None*, *location=None*, *cosmic=None*, *imgt=None*, *enaid=None*, *alias_symbol=None*, *alias_name=None*, *rgdid=None*, *omimid=None*, *ccdsid=None*, *lsdbs=None*, *ortholog_species=None*, *gene_family_identifier=None*, *limit=None*, *as_df=False*)

Method to query `pyhgnc.manager.models.Pmid`

Parameters

- **name** (*str or tuple(str) or None*) – HGNC approved name for the gene
- **symbol** (*str or tuple(str) or None*) – HGNC approved gene symbol
- **identifier** (*int or tuple(int) or None*) – HGNC ID. A unique ID created by the HGNC for every approved symbol
- **status** (*str or tuple(str) or None*) – Status of the symbol report, which can be either "Approved" or "Entry Withdrawn"
- **uuid** (*str or tuple(str) or None*) – universally unique identifier
- **locus_group** (*str or tuple(str) or None*) – group name for a set of related locus types as defined by the HGNC
- **orphanet** (*int ot tuple(int) or None*) – Orphanet database identifier (related to rare diseases and orphan drugs)
- **locus_type** (*str or tuple(str) or None*) – locus type as defined by the HGNC (e.g. RNA, transfer)
- **date_name_changed** (*str or tuple(str) or None*) – date the gene name was last changed (format: YYYY-mm-dd, e.g. 2017-09-29)
- **date_modified** (*str or tuple(str) or None*) – date the entry was last modified (format: YYYY-mm-dd, e.g. 2017-09-29)
- **date_symbol_changed** (*str or tuple(str) or None*) – date the gene symbol was last changed (format: YYYY-mm-dd, e.g. 2017-09-29)
- **date_approved_reserved** (*str or tuple(str) or None*) – date the entry was first approved (format: YYYY-mm-dd, e.g. 2017-09-29)
- **pubmedid** (*int ot tuple(int) or None*) – PubMed identifier
- **ensembl_gene** (*str or tuple(str) or None*) – Ensembl gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **horde** (*str or tuple(str) or None*) – symbol used within HORDE for the gene (not available in JSON)
- **vega** (*str or tuple(str) or None*) – Vega gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **lncrnadb** (*str or tuple(str) or None*) – Noncoding RNA Database identifier
- **uniprotid** (*str or tuple(str) or None*) – UniProt identifier

- **entrez** (*str or tuple(str) or None*) – Entrez gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **mirbase** (*str or tuple(str) or None*) – miRBase ID
- **iuphar** (*str or tuple(str) or None*) – The objectId used to link to the IUPHAR/BPS Guide to PHARMACOLOGY database
- **ucsc** (*str or tuple(str) or None*) – UCSC gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **snornabase** (*str or tuple(str) or None*) – snoRNABase ID
- **gene_family_name** (*int or tuple(int) or None*) – Gene family name
- **gene_family_identifier** – Gene family identifier
- **mgdid** (*int ot tuple(int) or None*) – Mouse Genome Database identifier
- **imgt** (*str or tuple(str) or None*) – Symbol used within international ImMunoGeneTics information system
- **enaid** (*str or tuple(str) or None*) – European Nucleotide Archive (ENA) identifier
- **alias_symbol** (*str or tuple(str) or None*) – Other symbols used to refer to a gene
- **alias_name** (*str or tuple(str) or None*) – Other names used to refer to a gene
- **pseudogeneorg** (*str or tuple(str) or None*) – Pseudogene.org ID
- **bioparadigmsslc** (*str or tuple(str) or None*) – Symbol used to link to the SLC tables database at bioparadigms.org for the gene
- **locationsortable** (*str or tuple(str) or None*) – locations sortable
- **ec_number** (*str or tuple(str) or None*) – Enzyme Commission number (EC number)
- **refseq_accession** (*str or tuple(str) or None*) – RefSeq nucleotide accession(s)
- **merops** (*str or tuple(str) or None*) – ID used to link to the MEROPS peptidase database
- **location** (*str or tuple(str) or None*) – Cytogenetic location of the gene (e.g. 2q34).
- **cosmic** (*str or tuple(str) or None*) – Symbol used within the Catalogue of somatic mutations in cancer for the gene
- **rgdid** (*int or tuple(int) or None*) – Rat genome database gene ID
- **omimid** (*int or tuple(int) or None*) – Online Mendelian Inheritance in Man (OMIM) ID
- **ccdsid** (*str or tuple(str) or None*) – Consensus CDS ID
- **lsdbs** (*str or tuple(str) or None*) – Locus Specific Mutation Database Name
- **ortholog_species** (*int or tuple(int) or None*) – Ortholog species NCBI taxonomy identifier

- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as `pandas.DataFrame`

Returns

- if *as_df* == *False* -> list(`models.Keyword`)
- if *as_df* == *True* -> `pandas.DataFrame`

Return type list[*models.HGNC*]

lsdb(*lsdb=None*, *url=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.LSDB* objects in database

Parameters

- **lsdb** (*str or tuple(str) or None*) – name(s) of the Locus Specific Mutation Database
- **url** (*str or tuple(str) or None*) – URL of the Locus Specific Mutation Database
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as `pandas.DataFrame`

Returns

- if *as_df* == *False* -> list(*models.LSDB*)
- if *as_df* == *True* -> `pandas.DataFrame`

Return type list(*models.LSDB*) or `pandas.DataFrame`

mgd(*mgdid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.MGD* objects in database

Parameters

- **mgdid** (*str or tuple(str) or None*) – Mouse genome informatics database ID(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit

- if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
- if limit == None -> all results

- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.MGD*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.MGD*) or pandas.DataFrame

omim(*omimid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)

Method to query *models.OMIM* objects in database

Parameters

- **omimid** (*str or tuple(str) or None*) – Online Mendelian Inheritance in Man (OMIM) ID(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.OMIM*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.OMIM*) or pandas.DataFrame

orthology_prediction(*ortholog_species=None*, *human_entrez_gene=None*, *human_ensembl_gene=None*, *human_name=None*, *human_symbol=None*, *human_chr=None*, *human_assert_ids=None*, *ortholog_species_entrez_gene=None*, *ortholog_species_ensembl_gene=None*, *ortholog_species_db_id=None*, *ortholog_species_name=None*, *ortholog_species_symbol=None*, *ortholog_species_chr=None*, *ortholog_species_assert_ids=None*, *support=None*, *hgnc_identifier=None*, *hgnc_symbol=None*, *limit=None*, *as_df=False*)

Method to query *pyhgnc.manager.models.OrthologyPrediction*

Parameters

- **ortholog_species** (*int*) – NCBI taxonomy identifier
- **human_entrez_gene** (*str*) – Entrez gene identifier
- **human_ensembl_gene** (*str*) – Ensembl identifier
- **human_name** (*str*) – human gene name
- **human_symbol** (*str*) – human gene symbol

- **human_chr** (*str*) – human chromosome
- **human_assert_ids** (*str*) –
- **ortholog_species_entrez_gene** (*str*) – Entrez gene identifier for ortholog
- **ortholog_species_ensembl_gene** (*str*) – Ensembl gene identifier for ortholog
- **ortholog_species_db_id** (*str*) – Species specific database identifier (e.g. MGI:1920453)
- **ortholog_species_name** (*str*) – gene name of ortholog
- **ortholog_species_symbol** (*str*) – gene symbol of ortholog
- **ortholog_species_chr** (*str*) – chromosome identifier (ortholog)
- **ortholog_species_assert_ids** (*str*) –
- **support** (*str*) –
- **hgnc_identifier** (*int*) – HGNC identifier
- **hgnc_symbol** (*str*) – HGNC symbol
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as `pandas.DataFrame`

Returns

- if *as_df* == *False* -> list(`models.Keyword`)
- if *as_df* == *True* -> `pandas.DataFrame`

Return type list(`models.Keyword`) or `pandas.DataFrame`

pubmed(*pubmedid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)

Method to query *models.PubMed* objects in database

Parameters

- **pubmedid** (*str or tuple(str) or None*) – alias symbol(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as `pandas.DataFrame`

Returns

- if *as_df* == *False* -> list(*models.PubMed*)
- if *as_df* == *True* -> `pandas.DataFrame`

Return type list(*models.PubMed*) or pandas.DataFrame

ref_seq(*accession=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.RefSeq* objects in database

Parameters

- **accession** (*str or tuple(str) or None*) – RefSeq accessionl(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.RefSeq*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.RefSeq*) or pandas.DataFrame

rgd(*rgdid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.RGD* objects in database

Parameters

- **rgdid** (*str or tuple(str) or None*) – Rat genome database gene ID(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)
- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as pandas.DataFrame

Returns

- if *as_df* == *False* -> list(*models.RGD*)
- if *as_df* == *True* -> pandas.DataFrame

Return type list(*models.RGD*) or pandas.DataFrame

uniprot(*uniprotid=None*, *hgnc_symbol=None*, *hgnc_identifier=None*, *limit=None*, *as_df=False*)
Method to query *models.UniProt* objects in database

Parameters

- **uniprotid** (*str or tuple(str) or None*) – UniProt identifier(s)
- **hgnc_symbol** (*str or tuple(str) or None*) – HGNC symbol(s)

- **hgnc_identifier** (*int or tuple(int) or None*) – identifiers(s) in *models.HGNC*
- **limit** (*int or tuple(int) or None*) –
  - if *isinstance(limit,int)==True* -> limit
  - if *isinstance(limit,tuple)==True* -> format:= tuple(page_number, results_per_page)
  - if limit == None -> all results
- **as_df** (*bool*) – if *True* results are returned as `pandas.DataFrame`

Returns

- if *as_df* == *False* -> list(*models.UniProt*)
- if *as_df* == *True* -> `pandas.DataFrame`

Return type list(*models.UniProt*) or `pandas.DataFrame`

CHAPTER 9

Data Models

PyHGNC uses SQLAlchemy to store the data in the database. You can use an instance of `pyhgnc.manager.query.QueryManager` to query the content of the database.

Entity–relationship model:

ToDo: Add ER figure here!

Contents

9.1 HGNC

class pyhgnc.manager.models.**HGNC**(***kwargs*)

Root class (table, model) for all other classes (tables, models) in PyHGNC. Basic information with 1:1 relationship to identifier are stored here

Warning:

- homeodb (Homeobox Database ID)
- horde_id (Symbol used within HORDE for the gene)

described in README, but not found in HGNC JSON file

Hint: To link to IUPHAR/BPS Guide to PHARMACOLOGY database only use the number (only use 1 from the result objectId:1)

Variables

- **name** (*str*) – HGNC approved name for the gene. Equates to the "APPROVED NAME" field within the gene symbol report
- **symbol** (*str*) – The HGNC approved gene symbol. Equates to the "APPROVED SYMBOL" field within the gene symbol report
- **orphanet** (*int*) – Orphanet ID
- **identifier** (*str*) – Unique ID created by the HGNC for every approved symbol (HGNC ID)
- **status** (*str*) – Status of the symbol report, which can be either "Approved" or "Entry Withdrawn"
- **uuid** (*str*) – universally unique identifier
- **locus_group** (*str*) – Group name for a set of related locus types as defined by the HGNC (e.g. non-coding RNA)
- **locus_type** (*str*) – Locus type as defined by the HGNC (e.g. RNA, transfer)
- **date_name_changed** (*date*) – date the gene name was last changed
- **date_modified** (*date*) – date the entry was last modified
- **date_symbol_changed** (*date*) – date the gene symbol was last changed
- **date_approved_reserved** (*date*) – date the entry was first approved

- **ensembl_gene** (*str*) – Ensembl gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **horde** (*str*) – symbol used within HORDE for the gene (not available in JSON)
- **vega** (*str*) – Vega gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **lncrnadb** (*str*) – Long Noncoding RNA Database identifier
- **entrez** (*str*) – Entrez gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **mirbase** (*str*) – miRBase ID
- **iuphar** (*str*) – The objectId used to link to the IUPHAR/BPS Guide to PHARMACOLOGY database
- **ucsc** (*str*) – UCSC gene ID. Found within the "GENE RESOURCES" section of the gene symbol report
- **snornabase** (*str*) – snoRNABase ID
- **imgt** (*str*) – Symbol used within international ImMunoGeneTics information system
- **pseudogeneorg** (*str*) – Pseudogene.org ID
- **bioparadigmsslc** (*str*) – Symbol used to link to the SLC tables database at bioparadigms.org for the gene
- **locationsortable** (*str*) – locations sortable
- **merops** (*str*) – ID used to link to the MEROPS peptidase database
- **location** (*str*) – Cytogenetic location of the gene (e.g. 2q34).
- **cosmic** (*str*) – Symbol used within the Catalogue of somatic mutations in cancer for the gene
- **rgds** (*list*) – relationship to *RGD*
- **omims** (*list*) – relationship to OMIM
- **ccdss** (*list*) – relationship to CCDS
- **lsdbs** (*list*) – relationship to LSDB
- **orthology_predictions** (*list*) – relationship to OrthologyPrediction
- **enzymes** (*list*) – relationship to Enzyme
- **gene_families** (*list*) – relationship to GeneFamily
- **refseq_accessions** (*list*) – relationship to RefSeq
- **mgds** (*list*) – relationship to MGD
- **uniprots** (*list*) – relationship to UniProt
- **pubmeds** (*list*) – relationship to PubMed
- **enas** (*list*) – relationship to ENA

9.2 AliasSymbol

class pyhgnc.manager.models.**AliasSymbol**(***kwargs*)

Other symbols used to refer to this gene as seen in the "SYNONYMS" field in the symbol report.

> **Attention:** Symbols previously approved by the HGNC for this gene are tagged with *is_previous_symbol==True*. Equates to the "PREVIOUS SYMBOLS & NAMES" field within the gene symbol report.

Variables

- ***alias_symbol*** (*str*) – other symbol
- **is_previous_symbol** (*bool*) – previously approved
- ***hgnc*** – back populates to *HGNC*

9.3 AliasName

class pyhgnc.manager.models.**AliasName**(***kwargs*)

Other names used to refer to this gene as seen in the "SYNONYMS" field in the gene symbol report.

> **Attention:** Gene names previously approved by the HGNC for this gene are tagged with *is_previous_name==True*.. Equates to the "PREVIOUS SYMBOLS & NAMES" field within the gene symbol report.

Variables

- ***alias_name*** (*str*) – other name
- **is_previous_name** (*bool*) – previously approved
- ***hgnc*** – back populates to *HGNC*

9.4 GeneFamily

class pyhgnc.manager.models.**GeneFamily**(***kwargs*)

Name and identifier given to a gene family or group the gene has been assigned to. Equates to the "GENE FAMILY" field within the gene symbol report.

Variables

- **familyid** (*int*) – family identifier
- **familyname** (*str*) – family name
- **hgncs** (*list*) – back populates to *HGNC*

9.5 RefSeq

class pyhgnc.manager.models.**RefSeq**(***kwargs*)

RefSeq nucleotide accession(s). Found within the"NUCLEOTIDE SEQUENCES" section of the gene symbol report.

See also RefSeq database for more information.

Variables

- **accession** (*str*) – RefSeq accession number
- **hgncs** (*list*) – back populates to *HGNC*

9.6 RGD

class pyhgnc.manager.models.**RGD**(***kwargs*)

Rat genome database gene ID. Found within the "HOMOLOGS" section of the gene symbol report

Variables

- **rgdid** (*str*) – Rat genome database gene ID
- **hgncs** – back populates to *HGNC*

9.7 OMIM

class pyhgnc.manager.models.**OMIM**(***kwargs*)

Online Mendelian Inheritance in Man (OMIM) ID

Variables

- **omimid** (*str*) – OMIM ID
- ***hgnc*** – back populates to *pyhgnc.manager.models.HGNC*

9.8 MGD

class pyhgnc.manager.models.**MGD**(***kwargs*)

Mouse genome informatics database ID. Found within the "HOMOLOGS" section of the gene symbol report

Variables

- **mgdid** (*str*) – Mouse genome informatics database ID
- **hgncs** (*list*) – back populates to *HGNC*

9.9 UniProt

class pyhgnc.manager.models.**UniProt**(***kwargs*)

Universal Protein Resource (UniProt) protein accession. Found within the "PROTEIN RESOURCES" section of the gene symbol report.

See also UniProt webpage for more information.

Variables

- **uniprotid** (*str*) – UniProt identifier
- **hgncs** (*list*) – back populates to *HGNC*

9.10 CCDS

class pyhgnc.manager.models.**CCDS**(***kwargs*)

Consensus CDS ID. Found within the "NUCLEOTIDE SEQUENCES" section of the gene symbol report.

See also CCDS for more information.

Variables

- **ccdsid** (*str*) – CCDS identifier
- ***hgnc*** – back populates to *HGNC*

9.11 PubMed

class pyhgnc.manager.models.**PubMed**(***kwargs*)

PubMed and Europe PubMed Central PMID

Variables

- **pubmedid** (*str*) – Pubmed identifier
- **hgncs** (*list*) – back populates to *HGNC*

9.12 ENA

class pyhgnc.manager.models.**ENA**(***kwargs*)

International Nucleotide Sequence Database Collaboration (GenBank, ENA and DDBJ) accession number(s). Found within the "NUCLEOTIDE SEQUENCES" section of the gene symbol report.

Variables

- **enaid** (*str*) – European Nucleotide Archive (ENA) identifier
- **hgncs** (*list*) – back populates to *HGNC*

9.13 Enzyme

class pyhgnc.manager.models.**Enzyme**(***kwargs*)

Enzyme Commission number (EC number)

Variables

- **ec_number** (*str*) – EC number
- **hgncs** (*list*) – back populates to *HGNC*

9.14 LSDB

class pyhgnc.manager.models.**LSDB**(***kwargs*)
The name of the Locus Specific Mutation Database and URL

Variables

- ***lsdb*** (*str*) – name of the Locus Specific Mutation Database
- **url** (*str*) – URL to database
- ***hgnc*** – back populates to *HGNC*

9.15 OrthologyPrediction

class pyhgnc.manager.models.**OrthologyPrediction**(***kwargs*)
Orthology Predictions

> **Warning:** OrthologyPrediction is still not correctly normalized and documented.

Variables

- **ortholog_species** (*int*) – NCBI taxonomy identifier
- **human_entrez_gene** (*int*) – Human Entrey gene identifier
- **human_ensembl_gene** (*str*) – Human Ensembl gene identifier
- **human_name** (*str*) – Human gene name
- **human_symbol** (*str*) – Human gene symbol
- **human_chr** (*str*) – Human gene chromosome location
- **human_assert_ids** (*str*) –
- **ortholog_species_entrez_gene** (*str*) – Ortholog species Entrez gene identifier
- **ortholog_species_ensembl_gene** (*str*) – Ortholog species Ensembl gene identifier
- **ortholog_species_db_id** (*str*) – Ortholog species database identifier
- **ortholog_species_name** (*str*) – Ortholog species gene name
- **ortholog_species_symbol** (*str*) – Ortholog species gene symbol
- **ortholog_species_chr** (*str*) – Ortholog species gene chromosome location
- **ortholog_species_assert_ids** (*str*) –
- **support** (*str*) –
- ***hgnc*** – back populates to *HGNC*

CHAPTER 10

Database functions

10.1 set_connection

pyhgnc.manager.database.**set_connection**(*connection='sqlite:////home/docs/.pyhgnc/data/pyhgnc.db'*)

Set the connection string for sqlalchemy and write it to the config file.

```
import pyhgnc
pyhgnc.set_connection('mysql+pymysql://{user}:{passwd}@{host}/{db}?charset=
↪{charset}')
```

Hint: valid connection strings

- mysql+pymysql://user:passwd@localhost/database?charset=utf8
- postgresql://scott:tiger@localhost/mydatabase
- mssql+pyodbc://user:passwd@database
- oracle://user:passwd@127.0.0.1:1521/database
- Linux: sqlite:////absolute/path/to/database.db
- Windows: sqlite:///C:path odatabase.db

Parameters **connection** (*str*) – sqlalchemy connection string

10.2 update

pyhgnc.manager.database.**update**(*connection=None*, *silent=False*, *hgnc_file_path=None*, *hcop_file_path=None*, *low_memory=False*)

Update the database with current version of HGNC

Parameters

- **connection** (*str*) – conncetion string
- **silent** (*bool*) – silent while import
- **hgnc_file_path** (*str*) – import from path HGNC
- **hcop_file_path** (*str*) – import from path HCOP (orthologs)
- **low_memory** (*bool*) – set to *True* if you have low memory

Returns

10.3 set_mysql_connection

pyhgnc.manager.database.**set_mysql_connection**(*host='localhost'*, *user='pyhgnc_user'*, *passwd='pyhgnc_passwd'*, *db='pyhgnc'*, *charset='utf8'*)

Method to set a MySQL connection

Parameters

- **host** (*str*) – MySQL database host
- **user** (*str*) – MySQL database user
- **passwd** (*str*) – MySQL database password
- **db** (*str*) – MySQL database name
- **charset** (*str*) – MySQL database charater set

Returns connection string

Return type str

CHAPTER 11

Roadmap

Next steps:

- Export of query results to different formats
- Tests for all query functions
- Improve documentation and tutorials
- Increase code coverage
- Collections of Jupyter notebooks with examples

CHAPTER 12

Technology

Warning: The following is in the moment not implemented! But already written here that a lot of things all still need to be done.

This page is meant to describe the development stack for PyHGNC, and should be a useful introduction for contributors.

12.1 Versioning

PyHGNC is kept under version control on GitHub and GitLab. This allows for changes in the software to be tracked over time, and for tight integration of the management aspect of software development. Code will be in future produced following the Git Flow philosophy, which means that new features are coded in branches off of the development branch and merged after they are triaged. Finally, develop is merged into master for releases. If there are bugs in releases that need to be fixed quickly, "hot fix" branches from master can be made, then merged back to master and develop after fixing the problem.

12.2 Testing in PyHGNC

PyHGNC is written with unit testing. Whenever possible, PyHGNC will prefer to practice test-driven development. This means that new ideas for functions and features are encoded as blank classes/functions and directly writing tests for the desired output. After tests have been written that define how the code should work, the functionality can be implemented.

Test-driven development requires us to think about design before making quick and dirty implementations. This results in better code. Additionally, thorough testing suites make it possible to catch when changes break existing functionality.

Tests are written with the standard `unittest` library.

12.2.1 Tox

While IDEs like PyCharm provide excellent testing tools, they are not programmatic. Tox is a python package that providesa CLI interface to run automated testing procedures (as well as other build functions, that aren't important to explain here). In PyHGNC, it is used to run the unit tests in the `tests` folder with the `py.test` harness. It also runs `check-manifest`, builds the documentation with `sphinx`, and computes the code coverage of the tests. The entire procedure is defined in `tox.ini`. Tox also allows test to be done on many different versions of Python.

12.2.2 Continuous Integration

Continuous integration is a philosophy of automatically testing code as it changes. PyHGNC makes use of the Travis CI server to perform testing because of its tight integration with GitHub. Travis automatically installs git hooks inside GitHub so it knows when a new commit is made. Upon each commit, Travis downloads the newest commit from GitHub and runs the tests configured in the `.travis.yml` file in the top level of the PyHGNC repository. This file effectively instructs the Travis CI server to run Tox. It also allows for the modification of the environment variables. This is used in PyHGNC to test on different versions of python.

12.2.3 Code Coverage

Is not implemented in the moment, but will be added in the next months.

12.3 Distribution

12.3.1 Versioning

PyHGNC tries to fulfil the following philosophy in future:

PyHGNC uses semantic versioning. In general, the project's version string will have a suffix `-dev` like in `0.3.4-dev` throughout the development cycle. After code is merged from feature branches to develop and it is time to deploy, this suffix is removed and develop branch is merged into master.

The version string appears in multiple places throughout the project, so BumpVersion is used to automate the updating of these version strings. See .bumpversion.cfg for more information.

12.3.2 Deployment

Code for PyHGNC is an open-source project on GitHub, but it is also distributed on the PyPI (pronounced Py-Pee-Eye) server. Travis CI has a wonderful integration with PyPI, so any time a tag is made on the master branch (and also assuming the tests pass), a new distribution is packed and sent to PyPI. Refer to the "deploy" section at the bottom of the `.travis.yml` file for more information, or the Travis CI PyPI deployment documentation. As a side note, Travis CI has an encryption tool so the password for the PyPI account can be displayed publicly on GitHub. Travis decrypts it before performing the upload to PyPI.

CHAPTER 13

Acknowledgment and contribution to scientific projects

Software development by:

- Christian Ebeling

The software development of PyHGNC by Fraunhofer Institute for Algorithms and Scientific Computing (SCAI) is supported and funded by the IMI (INNOVATIVE MEDICINES INITIATIVE) projects AETIONOMY and PHAGO. The aim of both projects is the identification of mechnisms in Alzhiemer's and Parkinson's disease in complex biological BEL networks for drug development.

CHAPTER 14

Indices and Tables

- genindex
- modindex
- search

Index

U