PyUniProt Documentation

Release 0.0.10

Christian Ebeling

Sep 08, 2020

Contents

for version: 0.0.10

`PyUniProt` is python software interface developed by the Department of Bioinformatics at the Fraunhofer Institute for Algorithms and Scientific Computing SCAI to the data provided by the European Bioinformatics Institute (EMBL-EBI) on their UniProt website.

The content of UniProt and the use of PyUniProt in combination with PyBEL supports successfully scientists in the IMI funded projects AETIONOMY and PHAGO in the identification of potential drugs in complex disease networks with several thousands of relationships compiled from BEL statements.

Aim of this software project is to provide an programmatic access to locally stored UniProt data and allow a filtered export in several formats used in the scientific community. Many query functions allow 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. PyUniProt is an ongoing project and needs improvement. We are happy if you want support our project or start a scientific cooperation with us.

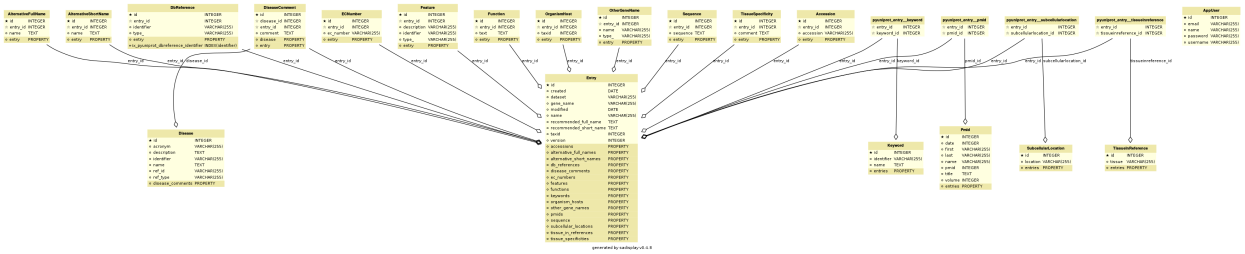


Fig. 1: ER model of PyUniProt database

- supported by IMI, AETIONOMY, PHAGO.

CHAPTER 1

Installation

1.1 System requirements

Because of the rich content of UniProt *PyUniProt* will create already for human, mouse and rat more than 5.7 million rows (08-04-017) with ~0.5 GiB of disk storage (depending on the used RDMS). Full installation (all organisms) will will need more than 5 GiB of disk storage.

Tests were performed on *Ubuntu 16.04, 4 x Intel Core i7-6560U CPU @ 2.20Ghz* with *16 GiB of RAM*. In general PyUniProt should work also on other systems like Windows, other Linux distributions or Mac OS.

1.2 Supported databases

PyUniProt uses SQLAlchemy to cover a wide spectrum of RDMSs (Relational database management system). 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. 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 a little bit different.

In general there are 2 ways to install the software:

1. Using stable version from pypi
2. Using latest development version from github

Please note that option number 2 is only recommended for experienced programmers interested in the source code. Also this software version is in development stage and we can not guarantee that the software is stable.

Often is make sense to avoid conflicts with other python installations by using different virtual environments. More information about an easy way to manage different virtual environments you find here.

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

```
sudo pip install pyuniprot
```

- If you have no sudo rights install as user

```
pip install --user pyuniprot
```

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

```
sudo python3 -m pip install pyuniprot
```

- If you are an experienced python with interest in the latest development version, clone and install from github

```
git clone https://github.com/cebel/pyuniprot.git
cd pyuniprot
pip install -e .
```

1.3.1 MySQL/MariaDB setup

In general you don't have to setup any database, because by default *pyuniprot* uses file based SQLite. 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 pyuniprot CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON pyuniprot.* TO 'pyuniprot_user'@'%' IDENTIFIED BY 'pyuniprot_
↪passwd';
FLUSH PRIVILEGES;
```

Start a python shell and set the MySQL configuration. If you have not changed anything in the SQL statements . . .

```
import pyuniprot
pyuniprot.set_mysql_connection()
```

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

```
import pyuniprot
pyuniprot.set_mysql_connection()
pyuniprot.set_mysql_connection(host='localhost', user='pyuniprot_user', passwd=
↪'pyuniprot_passwd', db='pyuniprot')
```

1.3.2 Updating

The updating process will download a gzipped file provided by the UniProt team on the download page

Please note that download file needs ~700 Mb of disk space and the update can take several hours (depending on your system). With every update a new database will created.

```
import pyuniprot
pyuniprot.update()
```

To make sure that the latest UniProt version is used, use the parameter *force_download*

```
import pyuniprot
pyuniprot.update(force_download=True)
```

1.4 Changing database configuration

Following functions allow to change the connection to you RDBMS (relational database management system). Next time you will use `pyuniprot` by default this connection will be used.

To set a new MySQL/MariaDB connection . . .

```
import pyuniprot
pyuniprot.set_mysql_connection(host='localhost', user='pyuniprot_user', passwd=
↪'pyuniprot_passwd', db='pyuniprot')
```

To set connection to other database systems use the *pyuniprot.set_connection* function.

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

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

CHAPTER 2

Quick start

This guide helps you to quickly setup your system in several minutes. But running the database import process and indexing takes still several hours.

Note: If your colleague have already executed the import process (perhaps on a special database server) please request the connection data to use PyUniProt without the need of running the update process.

Please make sure you have installed

1. MariaDB or any other supported RDMS *Supported databases*
2. Python3

Please note that you can also install with *pip* even if you are have no root rights on your machine. Just add *–user* behind *install*.

```
python3 -m pip install pyuniprot
```

Make sure that you have access to a database with user name and correct permissions. Otherwise execute on the MariaDB or MySQL console the following command as MySQL/MariaDb root. Replace user name, password and servername (here *localhost*) to our needs:

```
CREATE DATABASE `pyuniprot` CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'pyuniprot_user'@'localhost' IDENTIFIED BY 'pyuniprot_passwd';
GRANT ALL PRIVILEGES ON pytcd.* TO 'pyuniprot_user'@'localhost';
FLUSH PRIVILEGES;
```

Import UniProt data into database, but before change the SQLAlchemy connection string (line 2) to allow a connection to the database. If you have used the default code block and don't have to change anything.

Start your python console:

```
python3
```

Import the data:

```
import pyuniprot
pyuniprot.set_mysql_connection(host='localhost', user='pyuniprot_user', passwd=
↪'pyuniprot_passwd', db='pyuniprot')
pyuniprot.update(sqlalchemy_connection_string)
```

For examples how to query the database go to `pyuniprot.manager.database.Query` or *Tutorial*

CHAPTER 3

Tutorial

Here a short tutorial is planned.

CHAPTER 4

Query functions

Contents

- *feature*
- *function*
- *keyword*
- *ec_number*
- *subcellular_location*
- *tissue_specificity*
- *tissue_in_reference*

• *Query properties*

- *dbreference_types*
- *taxids*
- *datasets*
- *feature_types*
- *subcellular_locations*
- *tissues_in_references*
- *keywords*

4.1 Before you query

4.1.1 1. You can use % as a wildcard.

```
import pyuniprot
query = pyuniprot.query()

# exact search
query.entry(recommended_name='Amyloid beta A4 protein')

# starts with 'Amyloid beta'
query.entry(recommended_name='Amyloid beta%')

# ends with 'A4 protein'
query.entry(recommended_name='%A4 protein')

# contains 'beta A4'
query.entry(recommended_name='%beta A4%')
```

4.1.2 2. *limit* to restrict number of results

```
import pyuniprot
query = pyuniprot.query()

query.entry(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 pyuniprot
query = pyuniprot.query()

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

4.1.3 3. Return `pandas.DataFrame` as result

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

```
import pyuniprot
query = pyuniprot.query()

query.entry(as_df=True)
```

4.1.4 4. show all columns as dict

```
import pyuniprot
query = pyuniprot.query()

first_entry = query.entry(limit=1)[0]
first_entry.__dict__
```

4.1.5 5. Return single values with key name

```
import pyuniprot
query = pyuniprot.query()

query.entry(recommended_full_name='%kinase')[0].recommended_full_name
```

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

For example entry can access

- sequence
- accessions
- organism_hosts
- features
- functions
- ec_numbers
- db_references
- alternative_full_names

- alternative_short_names
- disease_comments
- tissue_specificities
- other_gene_names

```
import pyuniprot
query = pyuniprot.query()

r = query.entry(limit=1)[0]

r.sequence
r.accessions
r.organism_hosts
r.features
r.functions
r.ec_numbers
r.db_references
r.alternative_full_names
r.alternative_short_names
r.disease_comments
r.tissue_specificities
r.other_gene_names
```

But from EC number you can go back to entry

```
import pyuniprot
query = pyuniprot.query()

r = query.ec_number(ec_number='1.1.1.1')
[x.entry for x in r]
# following is crazy but possible, again go back to ec_number
[x.entry.ec_numbers for x in r]
```

4.1.7 7. Entry name is available in almost all methods

In almost all function you have the parameter *entry_name* (primary key for UniProt entries) even it is not part of the model.

```
import pyuniprot
query = pyuniprot.query()

query.other_gene_name(entry_name='A4_HUMAN')
```

4.2 entry

```
import pyuniprot
query = pyuniprot.query()

query.entry(name='1433E_HUMAN', recommended_full_name='14-3-3 protein epsilon', gene_
↪name='YWHAE')
```

Check documentation of `pyuniprot.manager.query.QueryManager.entry()` for all available parameters.

4.3 disease

```
import pyuniprot
query = pyuniprot.query()

query.disease(acronym='AD')
```

Check documentation of `pyuniprot.manager.query.QueryManager.disease()` for all available parameters.

4.4 disease_comment

```
import pyuniprot
query = pyuniprot.query()

query.disease_comment(comment='%Alzheimer%')
```

Check documentation of `pyuniprot.manager.query.QueryManager.disease_comment()` for all available parameters.

4.5 other_gene_name

```
import pyuniprot
query = pyuniprot.query()

query.other_gene_name(entry_name='A4_HUMAN')
```

Check documentation of `pyuniprot.manager.query.QueryManager.other_gene_name()` for all available parameters.

4.6 alternative_full_name

```
import pyuniprot
query = pyuniprot.query()

query.alternative_full_name(name='Alzheimer disease amyloid protein')
```

Check documentation of `pyuniprot.manager.query.QueryManager.alternative_full_name()` for all available parameters.

4.7 alternative_short_name

```
import pyuniprot
query = pyuniprot.query()

query.alternative_short_name(entry_name='A4_HUMAN')
```

Check documentation of `pyuniprot.manager.query.QueryManager.alternative_short_name()` for all available parameters.

4.8 accession

```
import pyuniprot
query = pyuniprot.query()

query.accession(accession='P05067', entry_name='A4_HUMAN')
```

Check documentation of `pyuniprot.manager.query.QueryManager.accession()` for all available parameters.

4.9 pmid

```
import pyuniprot
query = pyuniprot.query()

query.pmid(pmid=7644510)
```

Check documentation of `pyuniprot.manager.query.QueryManager.pmid()` for all available parameters.

4.10 organismHost

```
import pyuniprot
query = pyuniprot.query()

query.organism_host(taxid=9606)
# 0 results if you have only installed human
```

Check documentation of `pyuniprot.manager.query.QueryManager.organism_host()` for all available parameters.

4.11 dbReference

```
import pyuniprot
query = pyuniprot.query()

query.db_reference(type_='EMBL', identifier='U20972')
```

Check documentation of `pyuniprot.manager.query.QueryManager.db_reference()` for all available parameters.

4.12 feature

```
import pyuniprot
query = pyuniprot.query()

query.feature(type_='sequence variant', limit=1)
```

Check documentation of `pyuniprot.manager.query.QueryManager.feature()` for all available parameters.

4.13 function

```
import pyuniprot
query = pyuniprot.query()

query.function(text='%Alzheimer%')
```

Check documentation of `pyuniprot.manager.query.QueryManager.function()` for all available parameters.

4.14 keyword

```
import pyuniprot
query = pyuniprot.query()

r = query.keyword(name='Phagocytosis')[0]
[x.entry for x in r] # all proteins linked to keyword Phagocytosis
```

Check documentation of `pyuniprot.manager.query.QueryManager.keyword()` for all available parameters.

4.15 ec_number

```
import pyuniprot
query = pyuniprot.query()

query.ec_number(ec_number='1.1.1.1')
```

Check documentation of `pyuniprot.manager.query.QueryManager.ec_number()` for all available parameters.

4.16 subcellular_location

```
import pyuniprot
query = pyuniprot.query()

query.subcellular_location(location='Autophagosome lumen')
```

Check documentation of `pyuniprot.manager.query.QueryManager.subcellular_location()` for all available parameters.

4.17 tissue_specificity

```
import pyuniprot
query = pyuniprot.query()

query.tissue_specificity(comment='%brain%', limit=1)
```

Check documentation of `pyuniprot.manager.query.QueryManager.tissue_specificity()` for all available parameters.

4.18 tissue_in_reference

```
import pyuniprot
query = pyuniprot.query()

query.tissue_in_reference(tissue: 'Substantia nigra')
```

Check documentation of `pyuniprot.manager.query.QueryManager.tissue_in_reference()` for all available parameters.

CHAPTER 5

Query properties

5.1 dbreference_types

```
import pyuniprot
query = pyuniprot.query()
query.dbreference_types
```

5.2 taxids

```
import pyuniprot
query = pyuniprot.query()
query.taxids
```

5.3 datasets

```
import pyuniprot
query = pyuniprot.query()
query.datasets
```

5.4 feature_types

```
import pyuniprot
query = pyuniprot.query()
query.feature_types
```

5.5 subcellular_locations

```
import pyuniprot
query = pyuniprot.query()
query.subcellular_locations
```

5.6 tissues_in_references

```
import pyuniprot
query = pyuniprot.query()
query.tissues_in_references
```

5.7 keywords

```
import pyuniprot
query = pyuniprot.query()
query.keywords
```

CHAPTER 6

RESTful API

PyUniProt provides also a RESTful API web server.

Start the server with

```
pyuniprot web
```

Open PyUniProt Web API in a web browser.

CHAPTER 7

UniProt

We want to pay tribute to the UniProt team for their amazing resource their provide to the scientific community. `pyuniprot` only provides methods to download and locally query open accessible UniProt data.

7.1 About

Citation from UniProt website (about) [08/11/2017]: "The Universal Protein Resource (UniProt) is a comprehensive resource for protein sequence and annotation data. The UniProt databases are the UniProt Knowledgebase (UniProtKB), the UniProt Reference Clusters (UniRef), and the UniProt Archive (UniParc).

UniProt is a collaboration between the European Bioinformatics Institute (EMBL-EBI), the SIB Swiss Institute of Bioinformatics and the Protein Information Resource (PIR). Across the three institutes more than 100 people are involved through different tasks such as database curation, software development and support.

EMBL-EBI and SIB together used to produce Swiss-Prot and TrEMBL, while PIR produced the Protein Sequence Database (PIR-PSD). These two data sets coexisted with different protein sequence coverage and annotation priorities. TrEMBL (Translated EMBL Nucleotide Sequence Data Library) was originally created because sequence data was being generated at a pace that exceeded Swiss-Prot's ability to keep up. Meanwhile, PIR maintained the PIR-PSD and related databases, including iProClass, a database of protein sequences and curated families. In 2002 the three institutes decided to pool their resources and expertise and formed the UniProt consortium.

The UniProt consortium is headed by Alex Bateman (PI), Cathy Wu, and Ioannis Xenarios, supported by key staff, and receives valuable input from an independent Scientific Advisory Board."

Note: Please note that PyUniProt not covers all parts of UniProtKB. UniRef and UniParc are in the moment not acessible via the library. Only Swiss-Prot is included, TrEMBL will follow in the next version of PyUniProt.

7.2 Citation

Latest UniProt publication:

The UniProt Consortium UniProt: the universal protein knowledgebase Nucleic Acids Res. 45: D158-D169 (2017) (PDF)

7.3 Links

Link to data: UniProt ftp download page

Check the UniProt website for more information about data and online tools

CHAPTER 8

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

8.1 MySQL/MariaDB

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

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

User created with following command in MySQL/MariaDB:

```
GRANT ALL PRIVILEGES ON pyuniprot.* TO 'pyuniprot_user'@'%' IDENTIFIED BY 'pyuniprot_
↪passwd';
FLUSH PRIVILEGES;
```

Import of UniProt for human, mouse and rat (NCBI taxonomy IDs: 9606, 10090, 10116) data executed with:

```
import pyuniprot
pyuniprot.set_mysql_connection()
pyuniprot.update(taxids=[9606, 10090, 10116])
```

- CPU times: user 2h 5min 11s, sys: 35.8 s, total: 2h 5min 47s

CHAPTER 9

Query

Contents

9.1 Examples

For all string parameters you can use % as wildcard (please check the documentation below). All methods have a parameter `limit` which allows to limit the number of results and `as_df` which allows to return a *pandas.DataFrame*.

Initialize query object

```
import pyuniprot
pyuniprot.update(taxids=[9606,10090,10116]) # human, mouse, rat update
query = pyuniprot.query()
```

9.2 Methods by examples

search for . . .

human proteins with gene name 'TP53' (taxid=9606)

```
>>> query.entry(gene_name='TP53', taxid=9606)
[Cellular tumor antigen p53]
```

human proteins with *recommended full name* starts with 'Myeloid cell surface' (use % at the end)

```
>>> query.entry(recommended_full_name='Myeloid cell surface%', taxid=9606)
[Myeloid cell surface antigen CD33]
```

find all UniProt entries where the recommended full name contains 'CD33' (% at the start and end of search term) and return as *pandas.DataFrame*

```
>>> results = query.entry(name='%CD33%', taxid=9606, as_df=True)
# get first 2 lines of results with columns 'name','recommended_full_name', 'taxid'
>>> my_results_as_data_frame.ix[:2,('name','recommended_full_name', 'taxid')]
          name                      recommended_full_name  taxid
0   CD33_HUMAN          Myeloid cell surface antigen CD33   9606
1  CCD33_HUMAN  Coiled-coil domain-containing protein 33   9606
```

find entries by a list of gene names

```
>>> query.entry(name=('TREM2_HUMAN', 'CD33_HUMAN'))
[Myeloid cell surface antigen CD33, Triggering receptor expressed on myeloid cells 2]
```

If an attribute ends of an **s** it a clear hint that this is an 1:n or n:m relationship like keywords. There could be several proteins linked to a keyword, but also several keywords are linked to one protein. Next lines of code shows how to query for all proteins linked to the keyword 'Neurodegenaration' and returns the gene names.

```
>>> results = query.entry(keywords='Neurodegeneration')
>>> len(results) # number of results
322
>>> [x.gene_name for x in results][:3] # show only the first 2 gene names
['CHMP1A', 'CLN3', 'COQ8A']
```

Every element in the list represents a *pyuniprot.manager.models.Entry* instance:

```
>>> first_protein = results[0] # fetch first result
>>> type(first_protein)
pyuniprot.manager.models.Entry
>>> first_protein
Charged multivesicular body protein 1a
# get first 3 of all other keywords to this protein
>>> first_protein.keywords[:3]
[Reference proteome:KW-1185, Coiled coil:KW-0175, Repressor:KW-0678]
```

9.3 Properties

```
q.gene_forms
q.interaction_actions
q.actions
q.pathways
```

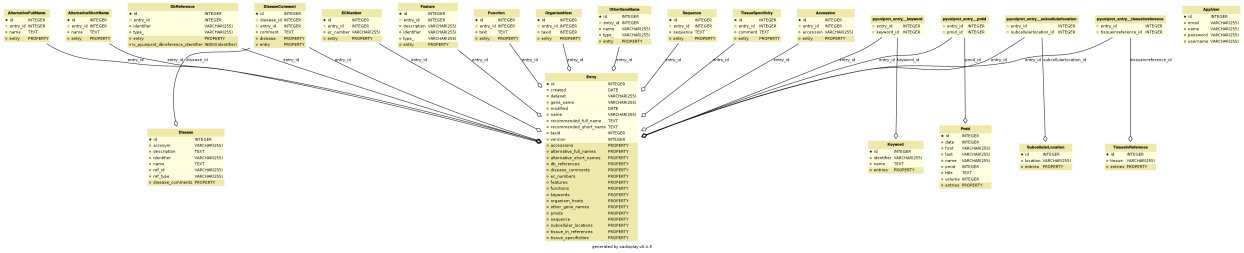
9.4 Query Manager Reference

CHAPTER 10

Data Models

PyUniProt uses SQLAlchemy to store the data in the database. Use instance of `pyuniprot.manager.query.QueryManager` to query the content of the database.

Entity–relationship model:



Contents

- *OrganismHost*
- *DbReference*
- *Feature*
- *Function*
- *Keyword*
- *ECNumber*
- *SubcellularLocation*
- *TissueSpecificity*
- *TissueInReference*

10.1 Entry

10.2 Accession

10.3 OtherGeneName

10.4 Sequence

10.5 Disease

10.6 DiseaseComment

10.7 AlternativeFullName

10.8 AlternativeShortName

10.9 Accession

10.10 Pmid

10.11 OrganismHost

10.12 DbReference

10.13 Feature

10.14 Function

10.15 Keyword

10.16 ECNumber

10.17 SubcellularLocation

10.18 TissueSpecificity

10.19 TissueInReference

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 PyUniProt, and should be a useful introduction for contributors.

12.1 Versioning

PyUniProt is kept under version control on GitHub. 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 PyUniProt

PyUniProt is written with unit testing. Whenever possible, PyUniProt will prefers 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 implementation can be written.

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 python package that provides a CLI interface to run automated testing procedures (as well as other build functions, that aren't important to explain here). In PyBEL, 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. PyUniProt 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 PyUniProt 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 PyUniProt to test many 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

PyUniProt tries to follow in future the following philosophy:

PyUniProt uses semantic versioning. In general, the project's version string will has 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 PyUniProt is open-source 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 PyUniProt 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