PyCTD Documentation

Release 0.5.10-dev

Christian Ebeling

Oct 06, 2022

Contents

for version: 0.5.10

`pyctd` is Python software developed by the Department of Bioinformatics at the Fraunhofer Institute for Algorithms and Scientific Computing (SCAI) to programmatically access and analyze data provided by the Comparative Toxicogenomics Database. For more information about CTD go to section CTD *About* .

The content of CTD and the use of PyCTD in combination with PyBEL facilitates scientists in the IMI funded projects AETIONOMY and PHAGO in the identification of potential drug targets in complex disease networks, which contain several thousands of relationships encoded as BEL statements.

The main aim of this software is to provide a programmatic access to locally stored CTD data and allow a filtered export in several formats used in the scientific community. We also focus our software development on the analysis and extension of biological disease knowledge networks. PyCTD is an ongoing project and needs further development as well as improvement. Please contact us, if you would like to support PyCTD or are interested in a scientific collaboration.

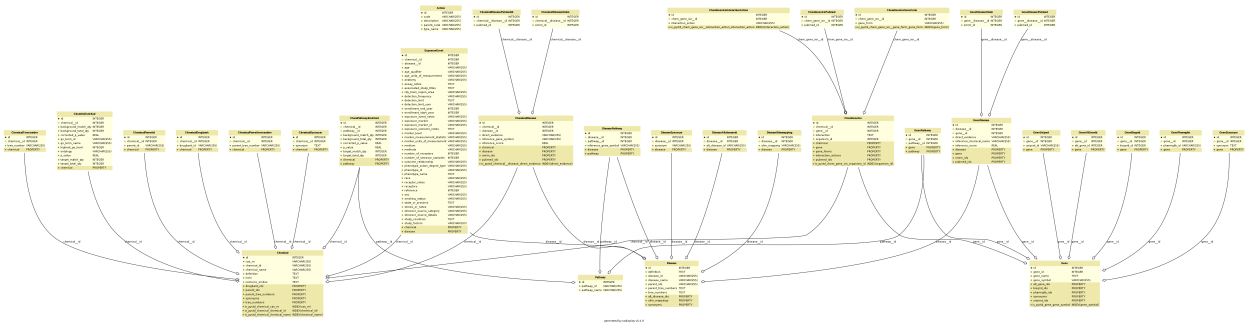

Fig. 1: ER model of pyctd database

- supported by IMI, AETIONOMY, PHAGO.

CHAPTER 1

Installation

1.1 System requirements

Because of the rich content of CTD *PyCTD* will create more than 230 million rows (04-28-017) with ~14 GiB 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 PyCTD should work also on other systems like Windows, other Linux distributions or Mac OS.

1.2 Supported Databases

PyCTD uses SQLAlchemy to cover a wide spectrum of RDMSs (relational database management system). We reccomend MySQL or MariaDB for best performance. If you cannot 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

`pyctd` provides a simple API so bioinformaticians and scientists with limited programming knowledge can easily use it to interface with CTD between chemical–gene/protein interactions, chemical–disease and gene–disease relationships.

1.3.1 Easiest

Download the latest stable code from PyPI with:

```
$ python3 -m pip install pyctd
```

1.3.2 Get the Latest

Download the most recent code from GitHub with:

```
$ python3 -m pip install git+https://github.com/pyctd/pyctd.git
```

1.3.3 For Developers

Clone the repository from GitHub and install in editable mode with:

```
$ git clone https://github.com/pyctd/pyctd.git
$ cd pyctd
$ python3 -m pip install -e .
```

1.4 Database Setup

1.4.1 MySQL/MariaDB setup

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

```
CREATE DATABASE pyctd CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON pyctd.* TO 'pyctd_user'@'%' IDENTIFIED BY 'pyctd_passwd';
FLUSH PRIVILEGES;
```

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

```
>>> import pyctd
>>> pyctd.set_mysql_connection()
```

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

```
>>> import pyctd
>>> pyctd.set_mysql_connection()
>>> pyctd.set_mysql_connection(host='localhost', user='pyctd_user', passwd='pyctd_
↪passwd', db='pyctd')
```

1.4.2 Updating

The updating process will download the files provided by the CTD on the download page

Warning: Please note the download needs 1.5 GB and the update takes ~2 hours (depending on your system)

```
>>> import pyctd
>>> pyctd.update()
```

1.5 Database Configuration

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

To set a new MySQL/MariaDB connection . . .

```
import pyctd
pyctd.set_mysql_connection()
pyctd.set_mysql_connection(host='localhost', user='pyctd_user', password='pyctd_passwd
↪', db='pyctd')
```

To set connection to other database systems use the *pyctd.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 pyctd
pyctd.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 PyCTD 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 pyctd
```

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

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

Import CTD 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 pyctd
>>> sqlalchemy_connection_string = 'mysql+pymysql://db_user:db_pwd@server_name/db_
↪name?charset=utf8'
>>> pyctd.update(sqlalchemy_connection_string)
```

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

CHAPTER 3

Comparative Toxicogenomics Database

`pyctd` only provides methods to download and locally query open accessible CTD data. We want to pay tribute to the following institutions for their amazing resource their provide to the scientific community:

1. Department of Biological Sciences, North Carolina State University
2. Department of Bioinformatics, The Mount Desert Island Biological Laboratory
3. Center for Human Health and the Environment, North Carolina State University

3.1 About

Citation from CTD website (about) [04/27/2017]: "*CTD is a robust, publicly available database that aims to advance understanding about how environmental exposures affect human health. It provides manually curated information about chemical–gene/protein interactions, chemical–disease and gene–disease relationships. These data are integrated with functional and pathway data to aid in development of hypotheses about the mechanisms underlying environmentally influenced diseases.*"

3.2 Links

Latest CTD publication:

The Comparative Toxicogenomics Database: update 2017; Nucleic Acids Res. 2017 Jan 4; 45(Database issue): D972–D978.; Published online 2016 Sep 19. doi: 10.1093/nar/gkw838; authors: Allan Peter Davis, Cynthia J. Grondin, Robin J. Johnson, Daniela Sciaky, Benjamin L. King, Roy McMorran, Jolene Wiegers, Thomas C. Wiegers, and Carolyn J. Mattingly; PubMed Central (PubReader, ePub (beta), PDF)

Link to data: CTD download page

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

CHAPTER 4

Query

4.1 Examples

For most of the 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*.

4.1.1 Methods

```
>>> import pyctd
>>> q = pyctd.query()
>>> q.get_diseases(disease_id='MESH:D000544', definition='%degenerative%')
>>> q.get_genes(gene_symbol='TSP_15922', uniprot_id='E5T972')
>>> q.get_pathways(pathway_name='%bla')
>>> q.get_chemicals(chemical_name='Alz%')
>>> q.get_chem_gene_interaction_action(organism_id='9606', gene_symbol='APP')
>>> q.get_gene__diseases(limit=10)
```

4.1.2 Properties

```
>>> import pyctd
>>> q = pyctd.query()
>>> q.gene_forms
>>> q.interaction_actions
>>> q.actions
>>> q.pathways
```

4.2 Query Manager Reference

CHAPTER 5

Data Manager Reference

5.1 Database Manager

5.2 Database Models

Not all database models are documented here in order to keep the documentation simple. In general `Query` should be used to query the content of the database

CHAPTER 6

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

6.1 MySQL/MariaDB

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

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

User created with following command in MySQL/MariaDB:

```
GRANT ALL PRIVILEGES ON pyctd.* TO 'pyctd_user'@'%' IDENTIFIED BY 'pyctd_passwd';
FLUSH PRIVILEGES;
```

Import of CTD data executed with:

```
import pyctd
pyctd.set_mysql_connection()
pyctd.update()
```

- CPU times: user 2h 2min 20s, sys: 37.7 s, total: 2h 2min 58s

CHAPTER 7

Roadmap

Next steps:

- Functions to identify potential drugs in BEL disease pathways
- mapping of interaction_action CTD and BEL relationships
- flask restful API
- Implement more query functions
- Export of query results to different formats
- Test for all supported *Supported Databases*
- Improve documentation and tutorials
- Increase code coverage
- Collections of Jupyter notebooks with examples

CHAPTER 8

Technology

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

8.1 Versioning

PyCTD 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.

8.2 Testing in PyCTD

PyCTD is written with unit testing. Whenever possible, PyCTD 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.

8.2.1 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.

8.2.2 Continuous Integration

Continuous integration is a philosophy of automatically testing code as it changes. PyCTD 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 PyCTD 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 PyCTD to test many different versions of python.

8.2.3 Code Coverage

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

8.3 Distribution

8.3.1 Versioning

PyCTD tries to follow in future the following philosophy:

PyCTD 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.

8.3.2 Deployment

Code for PyCTD 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 9

Acknowledgment and contribution to scientific projects

Software development by:

- Christian Ebeling
- Andrej Kontopez
- Charles Hoyt

The software development of PyCTD at 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 mechanisms in Alzheimer's and Parkinson's disease for drug development through creation and analysis of complex biological BEL networks.

CHAPTER 10

Indices and Tables

- genindex
- modindex
- search