PyCTD Documentation

Release 0.5.9

Christian Ebeling

Jul 31, 2017

Contents

1 Installation **3**
- 1.1 System requirements . . . 3
- 1.2 Supported Databases . . . 3
- 1.3 Install Software . . . 4
- 1.4 Database Setup . . . 4
- 1.5 Database Configuration . . . 5

2 Quick start **7**

3 Comparative Toxicogenomics Database **9**
- 3.1 About . . . 9
- 3.2 Links . . . 9

4 Query **11**
- 4.1 Examples . . . 11
- 4.2 Query Manager Reference . . . 12

5 Data Manager Reference **19**
- 5.1 Database Manager . . . 19
- 5.2 Database Models . . . 21

6 Benchmarks **35**
- 6.1 MySQL/MariaDB . . . 35

7 Roadmap **37**

8 Technology **39**
- 8.1 Versioning . . . 39
- 8.2 Testing in PyCTD . . . 39
- 8.3 Distribution . . . 40

9 Acknowledgment and contribution to scientific projects **41**

10 Indices and Tables **43**

Python Module Index **45**

for version: 0.5.9

`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.

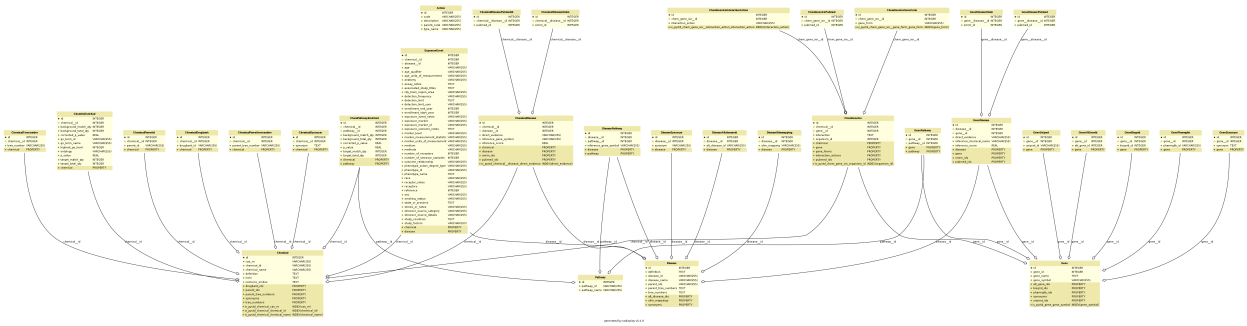


Fig. 1: ER model of pyctd database

- supported by IMI, AETIONOMY, PHAGO.

CHAPTER 1

Installation

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.

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

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.

Easiest

Download the latest stable code from PyPI with:

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

Get the Latest

Download the most recent code from GitHub with:

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

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 .
```

Database Setup

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')
```

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()
```

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

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.*"

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

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.

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)
```

Properties

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

Query Manager Reference

class pyctd.manager.query.**QueryManager**(*connection=None*, *echo=False*)

Query interface to database.

Parameters

- **connection** (*str*) – SQLAlchemy
- **echo** (*bool*) – True or False for SQL output of SQLAlchemy engine

actions

Gets the list of allowed actions

Return type list[str]

direct_evidences

Returns All available direct evidences for gene disease correlations

Return type list

gene_forms

Returns List of strings for all available gene forms

Return type list[str]

get_action(*limit=None*, *as_df=False*)

Parameters

- **limit** –
- **as_df** –

Returns

get_chem_gene_interaction_actions(*gene_name=None*, *gene_symbol=None*, *gene_id=None*, *limit=None*, *cas_rn=None*, *chemical_id=None*, *chemical_name=None*, *organism_id=None*, *interaction_sentence=None*, *chemical_definition=None*, *gene_form=None*, *interaction_action=None*, *as_df=False*)

Get all interactions for chemicals on a gene or biological entity (linked to this gene).

Chemicals can interact on different types of biological entities linked to a gene. A list of allowed entities linked to a gene can be retrieved via the attribute *gene_forms*.

Interactions are classified by a combination of interaction ('affects', 'decreases', 'increases') and actions ('activity', 'expression', ...). A complete list of all allowed interaction_actions can be retrieved via the attribute *interaction_actions*.

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **interaction_sentence** (*str*) – sentence describing the interactions
- **organism_id** (*int*) – NCBI TaxTree identifier. Example: 9606 for Human.
- **chemical_name** (*str*) – chemical name
- **chemical_id** (*str*) – chemical identifier
- **cas_rn** (*str*) – CAS registry number

- **chemical_definition** (*str*) –
- **gene_symbol** (*str*) – HGNC gene symbol
- **gene_name** (*str*) – gene name
- **gene_id** (*int*) – NCBI Entrez Gene identifier
- **gene_form** (*str*) – gene form
- **interaction_action** (*str*) – combination of interaction and actions
- **limit** (*int*) – maximum number of results

Return type list[*models.ChemGeneIxn*]

See also:

pyctd.manager.models.ChemGeneIxn

which is linked to: *pyctd.manager.models.Chemical* *pyctd.manager.models.Gene* *pyctd.manager.models.ChemGeneIxnPubmed*

Available interaction_actions and gene_forms `pyctd.manager.database.Query.interaction_actions()` `pyctd.manager.database.Query.gene_forms()`

get_chemical(*chemical_name=None*, *chemical_id=None*, *cas_rn=None*, *drugbank_id=None*, *parent_id=None*, *parent_tree_number=None*, *tree_number=None*, *synonym=None*, *limit=None*, *as_df=False*)

Get chemical

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **chemical_name** (*str*) – chemical name
- **chemical_id** (*str*) – cehmical identifier
- **cas_rn** (*str*) – CAS registry number
- **drugbank_id** (*str*) – DrugBank identifier
- **parent_id** (*str*) – identifiers of the parent terms
- **parent_tree_number** (*str*) – identifiers of the parent nodes
- **tree_number** (*str*) – identifiers of the chemical's nodes
- **synonym** (*str*) – chemical synonym
- **limit** (*int*) – maximum number of results

Returns list of *pyctd.manager.models.Chemical* objects

See also:

pyctd.manager.models.Chemical

get_chemical__by__disease(*disease_name*, *limit=None*, *as_df=False*)

Parameters

- **disease_name** –
- **limit** –
- **as_df** –

Returns

get_chemical_diseases(*direct_evidence=None*, *inference_gene_symbol=None*, *inference_score=None*, *inference_score_operator=None*, *cas_rn=None*, *chemical_name=None*, *chemical_id=None*, *chemical_definition=None*, *disease_definition=None*, *disease_id=None*, *disease_name=None*, *limit=None*, *as_df=False*)

Get chemical–disease associations with inference gene

Parameters

- **direct_evidence** – direct evidence
- **inference_gene_symbol** – inference gene symbol
- **inference_score** – inference score
- **inference_score_operator** – inference score operator
- **cas_rn** –
- **chemical_name** – chemical name
- **chemical_id** –
- **chemical_definition** –
- **disease_definition** –
- **disease_id** –
- **disease_name** – disease name
- **limit** (*int*) – maximum number of results
- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*

Returns list of `pyctd.manager.database.models.ChemicalDisease` objects

See also:

pyctd.manager.models.ChemicalDisease

which is linked to: *pyctd.manager.models.Disease* *pyctd.manager.models.Chemical*

get_disease(*disease_name=None*, *disease_id=None*, *definition=None*, *parent_ids=None*, *tree_numbers=None*, *parent_tree_numbers=None*, *slim_mapping=None*, *synonym=None*, *alt_disease_id=None*, *limit=None*, *as_df=False*)

Get diseases

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **limit** (*int*) – maximum number of results
- **disease_name** (*str*) – disease name
- **disease_id** (*str*) – disease identifier
- **definition** (*str*) – definition of disease
- **parent_ids** (*str*) – parent identifiers, delimiter |
- **tree_numbers** (*str*) – tree numbers, delimiter |
- **parent_tree_numbers** (*str*) – parent tree numbers, delimiter
- **slim_mapping** (*str*) – term derived from the MeSH tree structure for the "Diseases" [C] branch, that classifies MEDIC diseases into high-level categories

- **synonym** (*str*) – disease synonyms
- **alt_disease_id** (*str*) – alternative disease identifiers

Returns list of *pyctd.manager.models.Disease* object

See also:

pyctd.manager.models.Disease

Todo

normalize parent_ids, tree_numbers and parent_tree_numbers in *pyctd.manager.models.Disease*

---

get_disease_pathways(*disease_id=None*, *disease_name=None*, *pathway_id=None*, *pathway_name=None*, *disease_definition=None*, *limit=None*, *as_df=False*)

Get disease pathway link

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **disease_id** –
- **disease_name** –
- **pathway_id** –
- **pathway_name** –
- **disease_definition** –
- **limit** (*int*) – maximum number of results

Returns list of `pyctd.manager.database.models.DiseasePathway` objects

See also:

pyctd.manager.models.DiseasePathway

which is linked to: *pyctd.manager.models.Disease* *pyctd.manager.models.Pathway*

get_gene(*gene_name=None*, *gene_symbol=None*, *gene_id=None*, *synonym=None*, *uniprot_id=None*, *pharmgkb_id=None*, *biogrid_id=None*, *alt_gene_id=None*, *limit=None*, *as_df=False*)

Get genes

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **alt_gene_id** –
- **gene_name** (*str*) – gene name
- **gene_symbol** (*str*) – HGNC gene symbol
- **gene_id** (*int*) – NCBI Entrez Gene identifier
- **synonym** (*str*) – Synonym
- **uniprot_id** (*str*) – UniProt primary accession number
- **pharmgkb_id** (*str*) – PharmGKB identifier
- **biogrid_id** (*int*) – BioGRID identifier
- **limit** (*int*) – maximum of results

Return type list[*models.Gene*]

get_gene_disease(*direct_evidence=None*, *inference_chemical_name=None*, *inference_score=None*, *gene_name=None*, *gene_symbol=None*, *gene_id=None*, *disease_name=None*, *disease_id=None*, *disease_definition=None*, *limit=None*, *as_df=False*)

Get gene–disease associations

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **gene_id** (*int*) – gene identifier
- **gene_symbol** (*str*) – gene symbol
- **gene_name** (*str*) – gene name
- **direct_evidence** (*str*) – direct evidence
- **inference_chemical_name** (*str*) – inference_chemical_name
- **inference_score** (*float*) – inference score
- **inference_chemical_name** – chemical name
- **disease_name** – disease name
- **disease_id** – disease identifier
- **disease_definition** – disease definition
- **limit** (*int*) – maximum number of results

Returns list of `pyctd.manager.database.models.GeneDisease` objects

See also:

pyctd.manager.models.GeneDisease

which is linked to: *pyctd.manager.models.Chemical* *pyctd.manager.models.Gene*

get_gene_pathways(*gene_name=None*, *gene_symbol=None*, *gene_id=None*, *pathway_id=None*, *pathway_name=None*, *limit=None*, *as_df=False*)

Get gene pathway link

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **gene_name** (*str*) – gene name
- **gene_symbol** (*str*) – gene symbol
- **gene_id** (*int*) – NCBI Gene identifier
- **pathway_id** –
- **pathway_name** (*str*) – pathway name
- **limit** (*int*) – maximum number of results

Returns list of `pyctd.manager.database.models.GenePathway` objects

See also:

pyctd.manager.models.GenePathway

which is linked to: *pyctd.manager.models.Gene* *pyctd.manager.models.Pathway*

get_go_enriched__by__chemical_name(*chemical_name*, *limit=None*, *as_df=False*)

Parameters

- **chemical_name** –
- **limit** –
- **as_df** –

Returns

get_marker_chemical__by__disease_name(*disease_name*, *limit=None*, *as_df=False*)

Parameters

- **disease_name** –
- **limit** –
- **as_df** –

Returns

get_pathway(*pathway_name=None*, *pathway_id=None*, *limit=None*, *as_df=False*)

Get pathway

Note: Format of pathway_id is KEGG:X* or REACTOME:X* . X* stands for a sequence of digits

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **pathway_name** (*str*) – pathway name
- **pathway_id** (*str*) – KEGG or REACTOME identifier
- **limit** (*int*) – maximum number of results

Returns list of *pyctd.manager.models.Pathway* objects

See also:

pyctd.manager.models.Pathway

get_pathway_enriched__by__chemical_name(*chemical_name*, *limit=None*, *as_df=False*)

Parameters

- **chemical_name** –
- **limit** –
- **as_df** –

Returns

get_therapeutic_chemical__by__disease_name(*disease_name*, *limit=None*, *as_df=False*)

Get therapeutic chemical by disease name

Parameters

- **as_df** (*bool*) – if set to True result returns as *pandas.DataFrame*
- **limit** (*int*) – maximum number of results

- **disease_name** (*str*) – disease name

Returns therapeutic chemical

Return type list[*models.ChemicalDisease*]

interaction_actions

Returns List of strings for allowed interaction/actions combinations

Return type list[str]

pathways

Get all pathways

Return type list[*models.Pathway*]

CHAPTER 5

Data Manager Reference

Database Manager

class pyctd.manager.database.**DbManager**(*connection=None*)

Implements functions to upload CTD files into a database. Preferred SQL Alchemy database is MySQL with pymysql.

Parameters **connection** (*str*) – custom database connection SQL Alchemy string

db_import(*urls=None*, *force_download=False*)

Updates the CTD database

1.downloads all files from CTD

2.drops all tables in database

3.creates all tables in database

4.import all data from CTD files

Parameters

- **urls** (*iter[str]*) – An iterable of URL strings
- **force_download** (*bool*) – force method to download

Returns SQL Alchemy model instance, populated with data from URL

Return type models.Namespace

classmethod download_urls(*urls*, *force_download=False*)

Downloads all CTD URLs that don't exist

Parameters

- **urls** (*iter[str]*) – iterable of URL of CTD
- **force_download** (*bool*) – force method to download

static get_column_names_from_file(*file_path*)
returns column names from CTD download file

Parameters **file_path** (*str*) – path to CTD download file

static get_dtypes(*sqlalchemy_model*)

Parameters **sqlalchemy_model** –

Returns

classmethod get_index_and_columns_order(*columns_in_file_expected*, *columns_dict*, *file_path*)

Parameters

- **columns_in_file_expected** –
- **columns_dict** –
- **file_path** –

Returns

classmethod get_index_of_column(*column*, *file_path*)
Get index of a specific column name in a CTD file

Parameters

- **column** –
- **file_path** –

Returns int or None

classmethod get_path_to_file_from_url(*url*)
standard file path

Parameters **url** (*str*) – CTD download URL

import_one_to_many(*file_path*, *column_index*, *parent_table*, *column_in_one2many_table*)

Parameters

- **file_path** –
- **column_index** –
- **parent_table** –
- **column_in_one2many_table** –

Returns

import_table(*table*)
import table by Table object

Parameters **table** (*manager.table_conf.Table*) – Table object

import_table_in_db(*file_path*, *use_columns_with_index*, *column_names_in_db*, *table*)
Imports data from CTD file into database

Parameters

- **file_path** (*str*) – path to file
- **use_columns_with_index** (*list[int]*) – list of column indices in file
- **column_names_in_db** (*list[str]*) – list of column names (have to fit to models except domain_id column name)

- **table** – *manager.table.Table* object

import_tables(*only_tables=None*, *exclude_tables=None*)
Imports all data in database tables

Parameters

- **only_tables** (*set[str]*) – names of tables to be imported
- **exclude_tables** (*set[str]*) – names of tables to be excluded

mapper
returns a dictionary with keys of pyctd.manager.table_con.domains_to_map and pandas.DataFrame as values.

DataFrames column names:

•domain_id (represents the domain identifier of e.g. chemical)

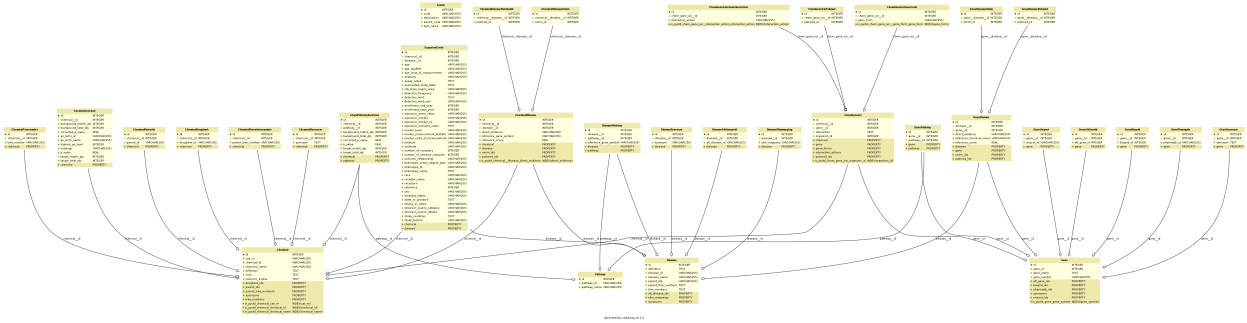
•domain__id (represents the primary key in domain table)

Returns dict of pandas DataFrames (keys:domain_name, values:DataFrame)

Return type dict of pandas.DataFrame

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 SQLAlchemy database models in this module describes all tables the database and fits the description in the table_conf module



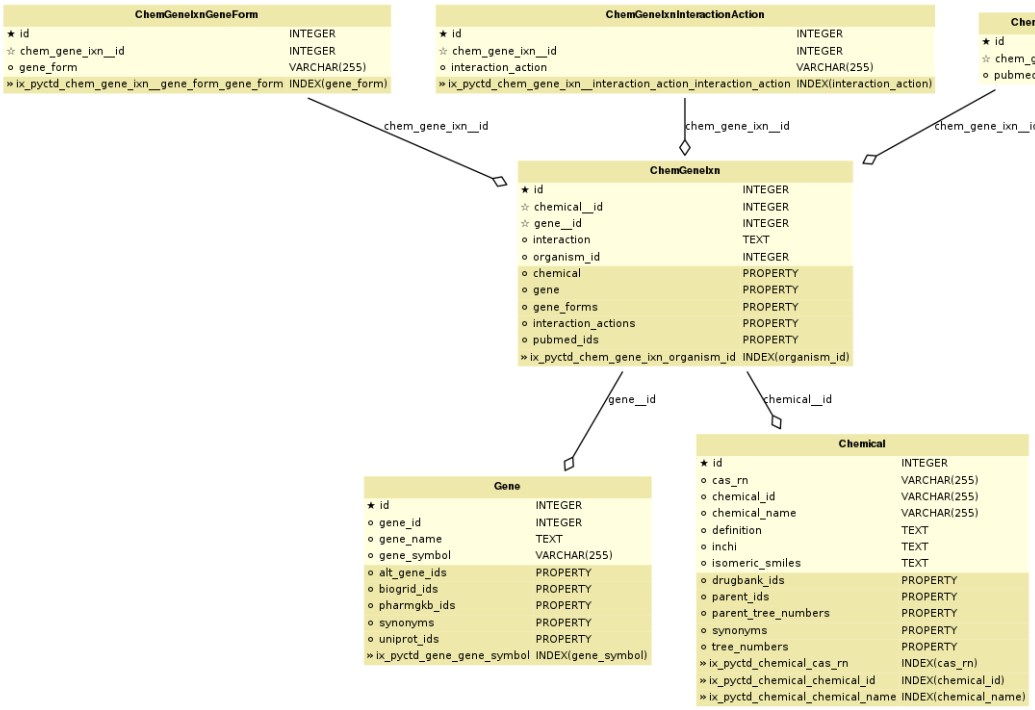
class pyctd.manager.models.**Action**(***kwargs*)
Chemical–gene interaction types

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemGeneIxn**(***kwargs*)



Chemical–gene interactions

reference:

- CTD Chemical–gene interactions

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`ChemGeneIxnGeneForm`**(***kwargs*)

Gene forms of Chemical–gene interactions

reference:

- CTD Gene forms of Chemical–gene interactions

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`ChemGeneIxnInteractionAction`**(***kwargs*)

Chemical–gene interactions actions

reference:

- CTD Chemical–gene interactions actions

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemGeneIxnPubmed**(**kwargs)
Chemical–gene interactions PubMed links

reference.

- CTD Chemical–gene interactions PubMed links

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemGoEnriched**(**kwargs)
Chemical–GO enriched associations

reference:

- CTD Chemical–GO enriched associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemPathwayEnriched**(**kwargs)
Chemical–pathway enriched associations

reference:

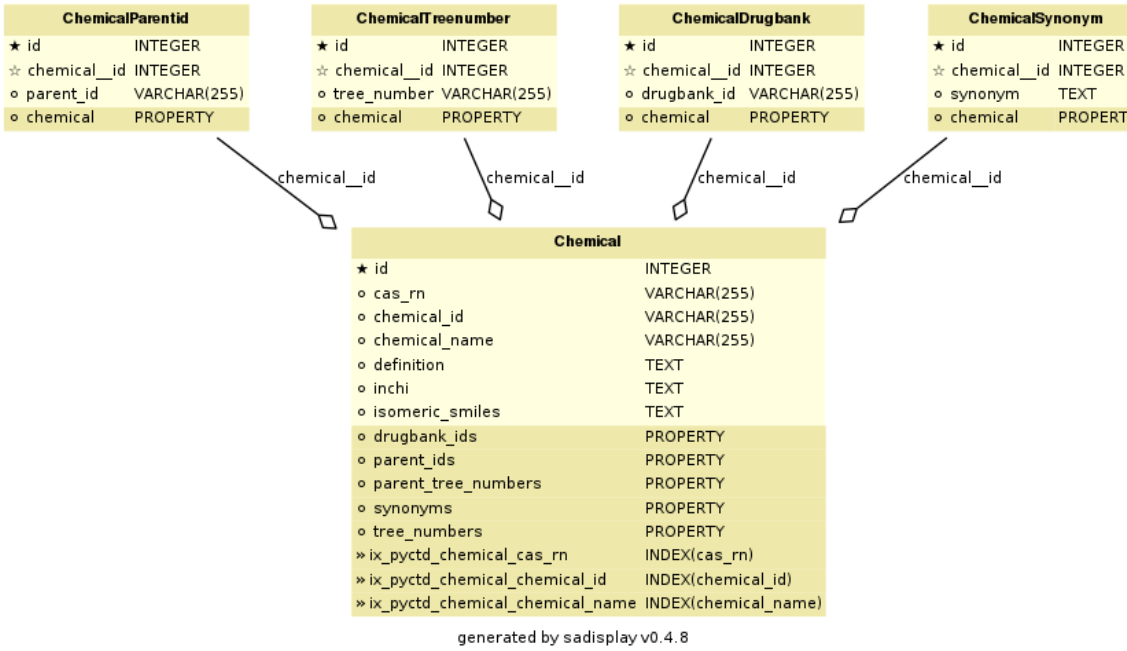
- CTD Chemical–pathway enriched associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**Chemical**(**kwargs)



Chemical vocabulary

reference:

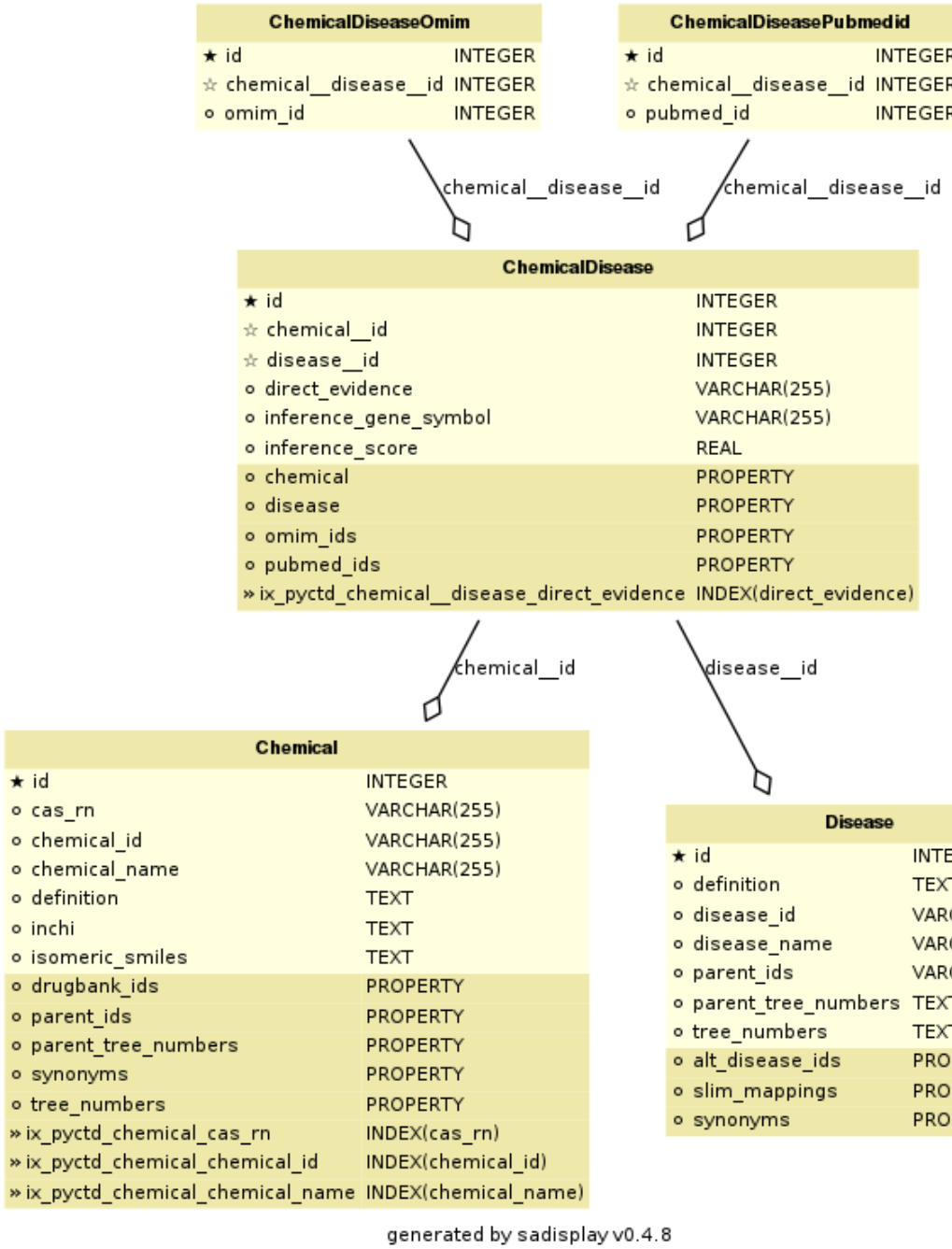
- CTD Help: Chemicals

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`ChemicalDisease`**(***kwargs*)



Chemical–disease associations

reference:

- CTD Chemical–disease associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`ChemicalDiseaseOmim`**(***kwargs*)

Online Mendelian Inheritance in Man (OMIM) mappings to Chemical–disease associations

reference:

- CTD OMIM

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemicalDiseasePubmedid**(***kwargs*)
PubMed Literature references to Chemical–disease associations

reference:

- CTD PubMed

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemicalDrugbank**(***kwargs*)
DrugBank identifiers to Chemical vocabulary

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemicalParentid**(***kwargs*)
Parent IDs of Chemical vocabulary

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemicalParenttreenumber**(***kwargs*)
Parent tree numbers of Chemical vocabulary

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ChemicalSynonym**(***kwargs*)
Synonymy to Chemical vocabulary

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

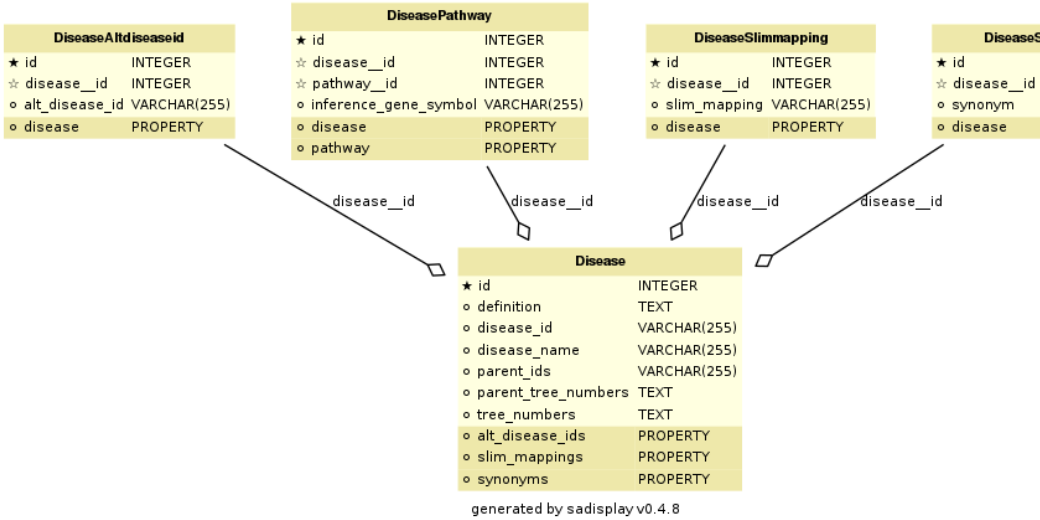
class pyctd.manager.models.**ChemicalTreenumber**(***kwargs*)
Tree numbers of Chemical vocabulary

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**Disease**(***kwargs*)



Disease vocabulary (MEDIC)

reference:

- CTD Disease vocabulary (MEDIC); structure
- CTD description disease

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

definition
definition of disease (str)

disease_id
MeSH or OMIM identifier (str)

disease_name
Disease name (str)

parent_ids
identifiers of the parent terms; 'l'-delimited list

parent_tree_numbers
identifiers of the parent nodes; 'l'-delimited list

tree_numbers
identifiers of the disease's nodes; 'l'-delimited list

class pyctd.manager.models.**DiseaseAltdiseaseid**(***kwargs*)
Alternative disease identifiers to Disease vocabulary (MEDIC)

reference:

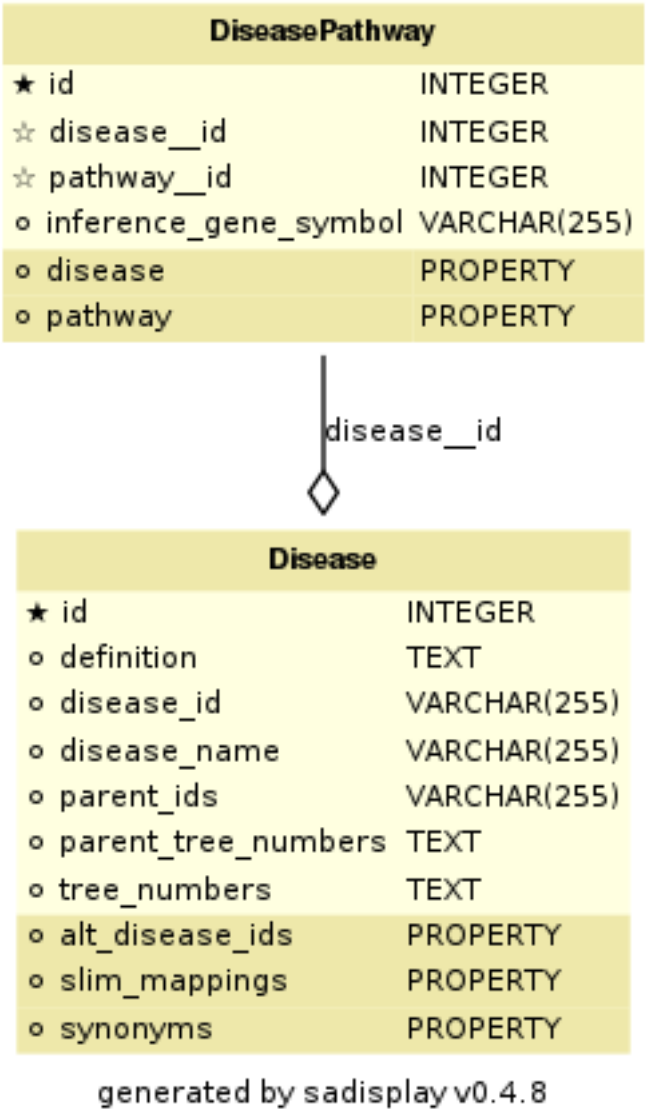
- CTD Disease (alternative identifier)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`DiseasePathway`**(***kwargs*)



Disease–pathway associations

reference:

- CTD Disease–pathway associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**DiseaseSlimmapping**(***kwargs*)
MEDIC-Slim mappings to Disease vocabulary (MEDIC)

reference:

- CTD MEDIC-Slim mappings

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**DiseaseSynonym**(***kwargs*)
Synonyms to Disease vocabulary (MEDIC)

reference:

- CTD Disease synonym

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**ExposureEvent**(***kwargs*)
Exposure–event associations

reference:

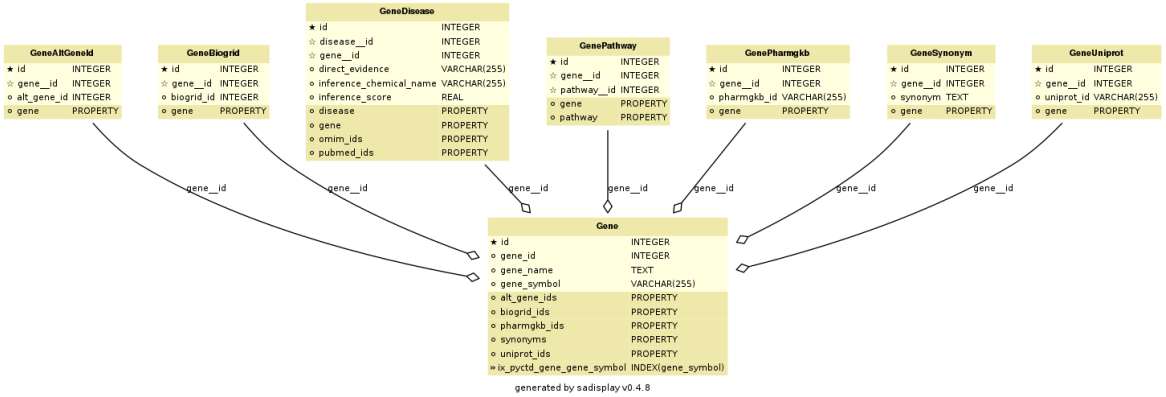
- CTD Exposure–event associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**Gene**(***kwargs*)



Gene vocabulary

reference:

- CTD Gene
- CTD Help: Genes

- UniProt accession numbers

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

alt_gene_ids
list of alternative NCBI Gene identifiers

biogrid_ids
list of BioGRID identifier

gene_id
NCBI Gene identifier

gene_name
gene name

gene_symbol
gene_symbol""

pharmgkb_ids
list of PharmGKB identifiers

synonyms
list of synonyms

uniprot_ids
UniProt accession number

class pyctd.manager.models.**GeneAltGeneId**(***kwargs*)
Alternative gene identifiers to Gene vocabulary

reference:

- CTD alternative NCBI Gene identifiers

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**GeneBiogrid**(***kwargs*)
BioGRID mappings to Gene vocabulary

reference:

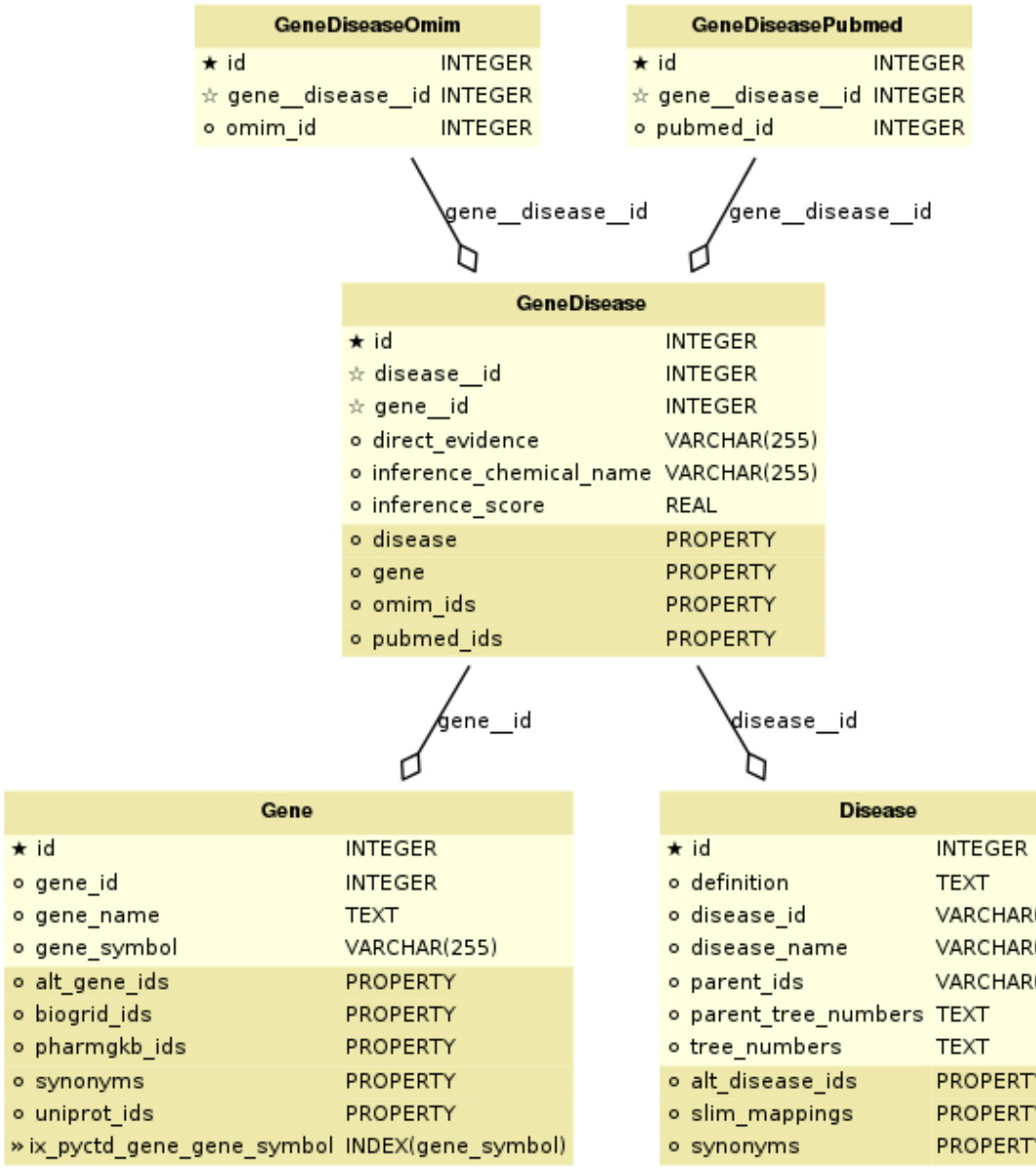
- CTD BioGRID
- BioGRID

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**GeneDisease**(***kwargs*)



Gene–disease associations

reference:

- CTD Gene–disease associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**GeneDiseaseOmim**(***kwargs*)

Online Mendelian Inheritance in Man (OMIM) mappings to Gene–disease associations

reference:

- CTD OMIM mappings to Gene–disease associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`GeneDiseasePubmed`**(***kwargs*)
PubMed references to Gene–disease associations

reference:

- CTD PubMed references to Gene–disease associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`GenePathway`**(***kwargs*)
Gene–pathway associations

reference:

- CTD Gene–pathway associations

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`GenePharmgkb`**(***kwargs*)
PharmGKB mapping to Gene vocabulary

reference:

- CTD PharmGKBIDs
- PharmGKB

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class `pyctd.manager.models.`**`GeneSynonym`**(***kwargs*)
Synonyms to Gene vocabulary

reference:

- CTD Synonyms

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**GeneUniprot**(***kwargs*)
UniProt mappings to Gene vocabulary

reference:

- CTD UniProt
- UniProt

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyctd.manager.models.**Pathway**(***kwargs*)
Pathway vocabulary

CTD link

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

pyctd.manager.models.**foreign_key_to**(*table_name*)
Creates a standard foreign key to a table in the database

Parameters **table_name** (*str*) – name of the table without TABLE_PREFIX

Returns foreign key column

Return type sqlalchemy.Column

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

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.

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.

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.

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.

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.

Code Coverage

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

Distribution

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.

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

Python Module Index

p

pyctd.manager.models, 21

Index

A

Action (class in pyctd.manager.models), 21
actions (pyctd.manager.query.QueryManager attribute), 12
alt_gene_ids (pyctd.manager.models.Gene attribute), 30

B

biogrid_ids (pyctd.manager.models.Gene attribute), 30

C

ChemGeneIxn (class in pyctd.manager.models), 21
ChemGeneIxnGeneForm (class in pyctd.manager.models), 22
ChemGeneIxnInteractionAction (class in pyctd.manager.models), 22
ChemGeneIxnPubmed (class in pyctd.manager.models), 22
ChemGoEnriched (class in pyctd.manager.models), 23
Chemical (class in pyctd.manager.models), 23
ChemicalDisease (class in pyctd.manager.models), 24
ChemicalDiseaseOmim (class in pyctd.manager.models), 25
ChemicalDiseasePubmedid (class in pyctd.manager.models), 26
ChemicalDrugbank (class in pyctd.manager.models), 26
ChemicalParentid (class in pyctd.manager.models), 26
ChemicalParenttreenumber (class in pyctd.manager.models), 26
ChemicalSynonym (class in pyctd.manager.models), 26
ChemicalTreenumber (class in pyctd.manager.models), 26
ChemPathwayEnriched (class in pyctd.manager.models), 23

D

db_import() (pyctd.manager.database.DbManager method), 19
DbManager (class in pyctd.manager.database), 19
definition (pyctd.manager.models.Disease attribute), 27
direct_evidences (pyctd.manager.query.QueryManager attribute), 12
Disease (class in pyctd.manager.models), 27
disease_id (pyctd.manager.models.Disease attribute), 27
disease_name (pyctd.manager.models.Disease attribute), 27
DiseaseAltdiseaseid (class in pyctd.manager.models), 27
DiseasePathway (class in pyctd.manager.models), 28
DiseaseSlimmapping (class in pyctd.manager.models), 29
DiseaseSynonym (class in pyctd.manager.models), 29
download_urls() (pyctd.manager.database.DbManager class method), 19

E

ExposureEvent (class in pyctd.manager.models), 29

F

foreign_key_to() (in module pyctd.manager.models), 33

G

Gene (class in pyctd.manager.models), 29
gene_forms (pyctd.manager.query.QueryManager attribute), 12
gene_id (pyctd.manager.models.Gene attribute), 30
gene_name (pyctd.manager.models.Gene attribute), 30
gene_symbol (pyctd.manager.models.Gene attribute), 30
GeneAltGeneId (class in pyctd.manager.models), 30
GeneBiogrid (class in pyctd.manager.models), 30
GeneDisease (class in pyctd.manager.models), 30
GeneDiseaseOmim (class in pyctd.manager.models), 31
GeneDiseasePubmed (class in pyctd.manager.models), 32
GenePathway (class in pyctd.manager.models), 32
GenePharmgkb (class in pyctd.manager.models), 32
GeneSynonym (class in pyctd.manager.models), 32
GeneUniprot (class in pyctd.manager.models), 32
get_action() (pyctd.manager.query.QueryManager method), 12
get_chem_gene_interaction_actions() (pyctd.manager.query.QueryManager method), 12

get_chemical() (pyctd.manager.query.QueryManager method), 13
get_chemical__by__disease() (pyctd.manager.query.QueryManager method), 13
get_chemical_diseases() (pyctd.manager.query.QueryManager method), 13
get_column_names_from_file() (pyctd.manager.database.DbManager static method), 19
get_disease() (pyctd.manager.query.QueryManager method), 14
get_disease_pathways() (pyctd.manager.query.QueryManager method), 15
get_dtypes() (pyctd.manager.database.DbManager static method), 20
get_gene() (pyctd.manager.query.QueryManager method), 15
get_gene_disease() (pyctd.manager.query.QueryManager method), 16
get_gene_pathways() (pyctd.manager.query.QueryManager method), 16
get_go_enriched__by__chemical_name() (pyctd.manager.query.QueryManager method), 16
get_index_and_columns_order() (pyctd.manager.database.DbManager class method), 20
get_index_of_column() (pyctd.manager.database.DbManager class method), 20
get_marker_chemical__by__disease_name() (pyctd.manager.query.QueryManager method), 17
get_path_to_file_from_url() (pyctd.manager.database.DbManager class method), 20
get_pathway() (pyctd.manager.query.QueryManager method), 17
get_pathway_enriched__by__chemical_name() (pyctd.manager.query.QueryManager method), 17
get_therapeutic_chemical__by__disease_name() (pyctd.manager.query.QueryManager method), 17

I

import_one_to_many() (pyctd.manager.database.DbManager method), 20
import_table() (pyctd.manager.database.DbManager method), 20
import_table_in_db() (pyctd.manager.database.DbManager method), 20
import_tables() (pyctd.manager.database.DbManager method), 21
interaction_actions (pyctd.manager.query.QueryManager attribute), 18

M

mapper (pyctd.manager.database.DbManager attribute), 21

P

parent_ids (pyctd.manager.models.Disease attribute), 27
parent_tree_numbers (pyctd.manager.models.Disease attribute), 27
Pathway (class in pyctd.manager.models), 33
pathways (pyctd.manager.query.QueryManager attribute), 18
pharmgkb_ids (pyctd.manager.models.Gene attribute), 30
pyctd.manager.models (module), 21

Q

QueryManager (class in pyctd.manager.query), 12

S

synonyms (pyctd.manager.models.Gene attribute), 30

T

tree_numbers (pyctd.manager.models.Disease attribute), 27

U

uniprot_ids (pyctd.manager.models.Gene attribute), 30