PubChemPy Documentation

Release 1.0.4

Matt Swain

Apr 11, 2017

Contents

PubChemPy provides a way to interact with PubChem in Python. It allows chemical searches by name, substructure and similarity, chemical standardization, conversion between chemical file formats, depiction and retrieval of chemical properties.

Here's a quick example showing how to search for a compound by name:

```
for compound in get_compounds('glucose', 'name'):
    print compound.cid
    print compound.isomeric_smiles
```

Here's how you get calculated properties for a specific compound:

```
vioxx = Compound.from_cid(5090)
print vioxx.molecular_formula
print vioxx.molecular_weight
print vioxx.xlogp
```

All the heavy lifting is done by PubChem's servers, using their database and chemical toolkits.

CHAPTER 1

Features

- Search PubChem Substance and Compound databases by name, SMILES, InChI and SDF.
- Retrieve the standardised Compound record for a given input structure.
- Convert between SDF, SMILES, InChI, PubChem CID and more.
- Retrieve calculated properties, fingerprints and descriptors.
- Generate 2D and 3D coordinates.
- Get IUPAC systematic names, trade names and all known synonyms for a given Compound.
- Download compound records as XML, ASNT/B, JSON, SDF and depiction as a PNG image.
- Construct property tables using *pandas* DataFrames.
- A complete Python wrapper around the PubChem PUG REST web service (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html).
- Supports Python versions 2.7 – 3.4.

CHAPTER 2

Useful links

- Source code is available on GitHub (https://github.com/mcs07/PubChemPy).
- Ask a question or report a bug on the Issue Tracker (https://github.com/mcs07/PubChemPy/issues).
- PUG REST API tutorial (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html) and documentation (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html).

CHAPTER 3

User guide

A step-by-step guide to getting started with PubChemPy.

Introduction

How PubChemPy works

PubChemPy relies entirely on the PubChem database and chemical toolkits provided via their PUG REST web service[1]. This service provides an interface for programs to automatically carry out the tasks that you might otherwise perform manually via the PubChem website (https://pubchem.ncbi.nlm.nih.gov).

This is important to remember when using PubChemPy: Every request you make is transmitted to the PubChem servers, evaluated, and then a response is sent back. There are some downsides to this: It is less suitable for confidential work, it requires a constant internet connection, and some tasks will be slower than if they were performed locally on your own computer. On the other hand, this means we have the vast resources of the PubChem database and chemical toolkits at our disposal. As a result, it is possible to do complex similarity and substructure searching against a database containing tens of millions of compounds in seconds, without needing any of the storage space or computational power on your own local computer.

[1] That's a lot of acronyms! PUG stands for "Power User Gateway", a term used to describe a variety of methods for programmatic access to PubChem data and services. REST stands for Representational State Transfer (https://en.wikipedia.org/wiki/Representational_state_transfer), which describes the specific architectural style of the web service.

The PUG REST web service

You don't need to worry too much about how the PubChem web service works, because PubChemPy handles all of the details for you. But if you want to go beyond the capabilities of PubChemPy, there is some helpful documentation on the PubChem website.

- PUG REST Tutorial (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST_Tutorial.html): Explains how the web service works with a variety of usage examples.
- PUG REST Specification (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html): A more comprehensive but dense specification that details every possible way to use the web service.

PubChemPy license

Installation

PubChemPy supports Python versions 2.7, 3.5, and 3.6. There are no other dependencies.

There are a variety of ways to download and install PubChemPy.

Option 1: Use pip (recommended)

The easiest and recommended way to install is using pip:

```
pip install pubchempy
```

This will download the latest version of PubChemPy, and place it in your *site-packages* folder so it is automatically available to all your python scripts.

If you don't already have pip installed, you can install it using get-pip.py (http://www.pip-installer.org/en/latest/installing.html):

```
curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
python get-pip.py
```

Option 2: Use conda

If you use Anaconda Python (https://www.continuum.io/anaconda-overview), install with conda:

```
conda install -c mcs07 pubchempy
```

Option 3: Download the latest release

Alternatively, download the latest release (https://github.com/mcs07/PubChemPy/releases/) manually and install yourself:

```
tar -xzvf PubChemPy-1.0.4.tar.gz
cd PubChemPy-1.0.4
python setup.py install
```

The setup.py command will install PubChemPy in your *site-packages* folder so it is automatically available to all your python scripts. Instead, you may prefer to just copy the pubchempy.py file into the desired project directory to only make it available to that project.

Option 4: Clone the repository

The latest development version of PubChemPy is always available on GitHub (https://github.com/mcs07/PubChemPy). This version is not guaranteed to be stable, but may include new features that have not yet been released. Simply clone the repository and install as usual:

```
git clone https://github.com/mcs07/PubChemPy.git
cd PubChemPy
python setup.py install
```

Getting started

This page gives a introduction on how to get started with PubChemPy. This assumes you already have PubChemPy *installed* (page 8).

Retrieving a Compound

Retrieving information about a specific Compound in the PubChem database is simple.

Begin by importing PubChemPy:

```
>>> import pubchempy as pcp
```

Let's get the Compound with CID 5090 (https://pubchem.ncbi.nlm.nih.gov/summary/summary.cgi?cid=5090):

```
>>> c = pcp.Compound.from_cid(5090)
```

Now we have a *Compound* (page 22) object called `c`. We can get all the information we need from this object:

```
>>> print c.molecular_formula
C17H14O4S
>>> print c.molecular_weight
314.35566
>>> print c.isomeric_smiles
CS(=O)(=O)C1=CC=C(C=C1)C2=C(C(=O)OC2)C3=CC=CC=C3
>>> print c.xlogp
2.3
>>> print c.iupac_name
3-(4-methylsulfonylphenyl)-4-phenyl-2H-furan-5-one
>>> print c.synonyms
[u'rofecoxib', u'Vioxx', u'Ceoxx', u'162011-90-7', u'MK 966', ... ]
```

Note: All the code examples in this documentation will assume you have imported PubChemPy as *pcp*. If you prefer, you can alternatively import specific functions and classes by name and use them directly:

```
from pubchempy import Compound, get_compounds
c = Compound.from_cid(1423)
cs = get_compounds('Aspirin', 'name')
```

Searching

What if you don't know the PubChem CID of the Compound you want? Just use the *get_compounds()* (page 21) function:

```
>>> results = pcp.get_compounds('Glucose', 'name')
>>> print results
[Compound(79025), Compound(5793), Compound(64689), Compound(206)]
```

The first argument is the identifier, and the second argument is the identifier type, which must be one of `name`, `smiles`, `sdf`, `inchi`, `inchikey` or `formula`. It looks like there are 4

compounds in the PubChem Database that have the name Glucose associated with them. Let's take a look at them in more detail:

```
>>> for compound in results:
...     print compound.isomeric_smiles
C([C@@H]1[C@H]([C@@H]([C@H]([C@H](O1)O)O)O)O)O
C([C@@H]1[C@H]([C@@H]([C@H](C(O1)O)O)O)O)O
C([C@@H]1[C@H]([C@@H]([C@H]([C@@H](O1)O)O)O)O)O
C(C1C(C(C(C(O1)O)O)O)O)O
```

It looks like they all have different stereochemistry information.

Retrieving the record for a SMILES string is just as easy:

```
>>> pcp.get_compounds('C1=CC2=C(C3=C(C=CC=N3)C=C2)N=C1', 'smiles')
[Compound(1318)]
```

Note: Beware that line notation inputs like SMILES and InChI can return automatically generated records that aren't actually present in PubChem, and therefore have no CID and are missing many properties that are too complicated to calculate on the fly.

That's all the most basic things you can do with PubChemPy. Read on for more some more advanced usage examples.

Searching

2D and 3D coordinates

By default, compounds are returned with 2D coordinates. Use the `record_type` keyword argument to specify otherwise:

```
pcp.get_compounds('Aspirin', 'name', record_type='3d')
```

Advanced search types

By default, requests look for an exact match with the input. Alternatively, you can specify substructure, superstructure, similarity and identity searches using the `searchtype` keyword argument:

```
pcp.get_compounds('CC', searchtype='superstructure', listkey_
↪count=3)
```

The `listkey_count` and `listkey_start` arguments can be used for pagination. Each `searchtype` has its own options that can be specified as keyword arguments. For example, similarity searches have a `Threshold`, and super/substructure searches have

`MatchIsotopes`. A full list of options is available in the PUG REST Specification (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html).

Note: These types of search are *slow*.

Getting a full results list for common compound names

For some very common names, PubChem maintains a filtered whitelist of human-chosen CIDs with the intention of reducing confusion about which is the 'right' result. In the past, a search for Glucose would return four different results, each with different stereochemistry information. But now, a single result is returned, which has been chosen as 'correct' by the PubChem team.

Unfortunately it isn't directly possible to return to the previous behaviour, but there is a straightforward workaround: Search for Substances with that name (which are completely unfiltered) and then get the compounds that are derived from those substances.

There area a few different ways you can do this using PubChemPy, but the easiest is probably using the `get_cids` function:

```
>>> pcp.get_cids('2-nonenal', 'name', 'substance', list_return='flat
↪')
[17166, 5283335, 5354833]
```

This searches the substance database for '2-nonenal', and gets the CID for the compound associated with each substance. By default, this returns a mapping between each SID and CID, but the `list_return='flat'` parameter flattens this into just a single list of unique CIDs.

You can then use `Compound.from_cid` to get the full Compound record, equivalent to what is returned by get_compounds:

```
>>> cids = pcp.get_cids('2-nonenal', 'name', 'substance', list_
↪return='flat')
>>> [pcp.Compound.from_cid(cid) for cid in cids]
[Compound(17166), Compound(5283335), Compound(5354833)]
```

Compound

The *get_compounds()* (page 21) function returns a list of *Compound* (page 22) objects. You can also instantiate a *Compound* (page 22) object directly if you know its CID:

```
c = pcp.Compound.from_cid(6819)
```

Dictionary representation

Each *Compound* (page 22) has a `record` property, which is a dictionary that contains the all the information about the compound, produced exactly from the JSON response from the PubChem API. All other properties are derived from this record.

Additionally, each *Compound* (page 22) provides a `to_dict()` method that returns PubChemPy's own dictionary representation of the Compound data. As well as being more concisely formatted than the raw `record`, this method also takes an optional parameter to filter the list of the desired properties:

```
>>> c = pcp.Compound.from_cid(962)
>>> c.to_dict(properties=['atoms', 'bonds', 'inchi'])
{'atoms': [{'aid': 1, 'element': 'o', 'x': 2.5369, 'y': -0.155},
           {'aid': 2, 'element': 'h', 'x': 3.0739, 'y': 0.155},
           {'aid': 3, 'element': 'h', 'x': 2, 'y': 0.155}],
 'bonds': [{'aid1': 1, 'aid2': 2, 'order': 'single'},
           {'aid1': 1, 'aid2': 3, 'order': 'single'}],
 'inchi': u'InChI=1S/H2O/h1H2'}
```

3D Compounds

Many properties are missing from 3D records, and the following properties are *only* available on 3D records:

- `volume_3d`
- `multipoles_3d`
- `conformer_rmsd_3d`
- `effective_rotor_count_3d`
- `pharmacophore_features_3d`
- `mmff94_partial_charges_3d`
- `mmff94_energy_3d`
- `conformer_id_3d`
- `shape_selfoverlap_3d`
- `feature_selfoverlap_3d`
- `shape_fingerprint_3d`

Substance

The PubChem Substance database contains all chemical records deposited in PubChem in their most raw form, before any significant processing is applied. As a result, it contains duplicates, mixtures, and some records that don't make chemical sense. This means that Substance records contain fewer calculated properties, however they do have additional information about the original source that deposited the record.

The PubChem Compound database is constructed from the Substance database using a standardization and deduplication process. Hence each Compound may be derived from a number of different Substances.

Retrieving substances

Retrieve Substances using the *get_substances()* (page 21) function:

```
>>> results = pcp.get_substances('Coumarin 343', 'name')
>>> print results
[Substance(24864499), Substance(85084977), Substance(126686397),␣
↪Substance(143491255), Substance(152243230), Substance(162092514),␣
↪Substance(162189467), Substance(186021999), Substance(206257050)]
```

You can also instantiate a Substance directly from its SID:

```
>>> substance = pcp.Substance.from_sid(223766453)
>>> print substance.synonyms
['2-(Acetyloxy)-benzoic acid', '2-(acetyloxy)benzoic acid', '2-
↪acetoxy benzoic acid', '2-acetoxy-benzoic acid', '2-
↪acetoxybenzoic acid', '2-acetyloxybenzoic acid', 'BSYNRYMUTXBXSQ-
↪UHFFFAOYSA-N', 'acetoxybenzoic acid', 'acetyl salicylic acid',
↪'acetyl-salicylic acid', 'acetylsalicylic acid', 'aspirin', 'o-
↪acetoxybenzoic acid']
>>> print substance.source_id
BSYNRYMUTXBXSQ-UHFFFAOYSA-N
>>> print substance.standardized_cid
2244
>>> print substance.standardized_compound
Compound(2244)
```

Properties

The `get_properties` function allows the retrieval of specific properties without having to deal with entire compound records. This is especially useful for retrieving the properties of a large number of compounds at once:

```
p = pcp.get_properties('IsomericSMILES', 'CC', 'smiles', searchtype=
↪'superstructure')
```

Multiple properties may be specified in a list, or in a comma-separated string. The available properties are: MolecularFormula, MolecularWeight, CanonicalSMILES, IsomericSMILES, InChI, InChIKey, IUPACName, XLogP, ExactMass, MonoisotopicMass, TPSA, Complexity, Charge, HBondDonorCount, HBondAcceptorCount, RotatableBondCount, HeavyAtomCount, IsotopeAtomCount, AtomStereoCount, DefinedAtomStereoCount, UndefinedAtomStereoCount, BondStereoCount, DefinedBondStereoCount, UndefinedBondStereoCount, CovalentUnitCount, Volume3D, XStericQuadrupole3D, YStericQuadrupole3D, ZStericQuadrupole3D, FeatureCount3D, FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D, FeatureHydrophobeCount3D, ConformerModelRMSD3D, EffectiveRotorCount3D, ConformerCount3D.

Synonyms

Get a list of synonyms for a given input using the `get_synonyms` function:

```
pcp.get_synonyms('Aspirin', 'name')
pcp.get_synonyms('Aspirin', 'name', 'substance')
```

Inputs that match more than one SID/CID will have multiple, separate synonyms lists returned.

Identifiers

There are three functions for getting a list of identifiers for a given input:

- `pcp.get_cids`
- `pcp.get_sids`
- `pcp.get_aids`

For example, passing a CID to get_sids will return a list of SIDs corresponding to the Substance records that were standardised and merged to produce the given Compound.

pandas integration

Getting *pandas*

pandas must be installed to use its functionality from within PubChemPy. The easiest way is to use pip:

```
pip install pandas
```

See the pandas documentation (http://pandas.pydata.org/pandas-docs/stable/) for more information.

Usage

It is possible for `get_compounds`, `get_substances` and `get_properties` to return a pandas DataFrame:

```
df1 = pcp.get_compounds('C20H41Br', 'formula', as_dataframe=True)
df2 = pcp.get_substances([1, 2, 3, 4], as_dataframe=True)
df3 = pcp.get_properties(['isomeric_smiles', 'xlogp', 'rotatable_
↪bond_count'], 'C20H41Br', 'formula', as_dataframe=True)
```

An existing list of Compound objects can be converted into a dataframe, optionally specifying the desired columns:

```
cs = pcp.get_compounds('C20H41Br', 'formula')
df4 = pcp.compounds_to_frame(cs, properties=['isomeric_smiles',
↪'xlogp', 'rotatable_bond_count'])
```

Download

The download function is for saving a file to disk. The following formats are available: XML, ASNT/B, JSON, SDF, CSV, PNG, TXT. Beware that not all formats are available for all types of information. SDF and PNG are only available for full Compound and Substance records, and CSV is best suited to tables of properties and identifiers.

Examples:

```
pcp.download('PNG', 'asp.png', 'Aspirin', 'name')
pcp.download('CSV', 's.csv', [1,2,3], operation='property/
↪CanonicalSMILES,IsomericSMILES')
```

For PNG images, the `image_size` argument can be used to specfiy `large`, `small` or `<width>x<height>`.

Advanced

Avoiding TimeoutError

If there are too many results for a request, you will receive a TimeoutError. There are different ways to avoid this, depending on what type of request you are doing.

If retrieving full compound or substance records, instead request a list of cids or sids for your input, and then request the full records for those identifiers individually or in small groups. For example:

```
sids = get_sids('Aspirin', 'name')
for sid in sids:
    s = Substance.from_sid(sid)
```

When using the `formula` namespace or a `searchtype`, you can also alternatively use the `listkey_count` and `listkey_start` keyword arguments to specify pagination. The `listkey_count` value specifies the number of results per page, and the `listkey_start` value specifies which page to return. For example:

```
get_compounds('CC', 'smiles', searchtype='substructure', listkey_
↪count=5)
get('C10H21N', 'formula', listkey_count=3, listkey_start=6)
```

Logging

PubChemPy can generate logging statements if required. Just set the desired logging level:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

The logger is named 'pubchempy'. There is more information on logging in the Python logging documentation (http://docs.python.org/2/howto/logging.html).

Using behind a proxy

When using PubChemPy behind a proxy, you may receive a `URLError`:

```
URLError: <urlopen error [Errno 65] No route to host>
```

A simple fix is to specify the proxy information via urllib. For Python 3:

```
import urllib
proxy_support = urllib.request.ProxyHandler({
    'http': 'http://<proxy.address>:<port>',
    'https': 'https://<proxy.address>:<port>'
})
opener = urllib.request.build_opener(proxy_support)
urllib.request.install_opener(opener)
```

For Python 2:

```
import urllib2
proxy_support = urllib2.ProxyHandler({
    'http': 'http://<proxy.address>:<port>',
    'https': 'https://<proxy.address>:<port>'
})
opener = urllib2.build_opener(proxy_support)
urllib2.install_opener(opener)
```

Custom requests

If you wish to perform more complicated requests, you can use the `request` function. This is an extremely simple wrapper around the REST API that allows you to construct any sort of request from a few parameters. The PUG REST Specification (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html) has all the information you will need to formulate your requests.

The `request` function simply returns the exact response from the PubChem server as a string. This can be parsed in different ways depending on the output format you choose. See the Python json (http://docs.python.org/2/library/json.html), xml (http://docs.python.org/2/library/xml.etree.elementtree.html) and csv

(http://docs.python.org/2/library/csv.html) packages for more information. Additionally, cheminformatics toolkits such as Open Babel (http://openbabel.org/docs/current/UseTheLibrary/Python.html) and RDKit (http://www.rdkit.org) offer tools for handling SDF files in Python.

The `get` function is very similar to the `request` function, except it handles `listkey` type responses automatically for you. This makes things simpler, however it means you can't take advantage of using the same `listkey` repeatedly to obtain different types of information. See the PUG REST specification (https://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html) for more information on how *listkey* responses work.

Summary of possible inputs

```
<identifier> = list of cid, sid, aid, source, inchikey, listkey;␣
↪string of name, smiles, xref, inchi, sdf;
<domain> = substance | compound | assay

compound domain
<namespace> = cid | name | smiles | inchi | sdf | inchikey |
↪<structure search> | <xref> | listkey | formula
<operation> = record | property/[comma-separated list of property␣
↪tags] | synonyms | sids | cids | aids | assaysummary |␣
↪classification

substance domain
<namespace> = sid | sourceid/<source name> | sourceall/<source name>
↪ | name | <xref> | listkey
<operation> = record | synonyms | sids | cids | aids | assaysummary␣
↪| classification

assay domain
<namespace> = aid | listkey | type/<assay type> | sourceall/<source␣
↪name>
<assay type> = all | confirmatory | doseresponse | onhold | panel |␣
↪rnai | screening | summary
<operation> = record | aids | sids | cids | description | targets/
↪{ProteinGI, ProteinName, GeneID, GeneSymbol} | doseresponse/sid

<structure search> = {substructure | superstructure | similarity |␣
↪identity}/{smiles | inchi | sdf | cid}
<xref> = xref/{RegistryID | RN | PubMedID | MMDBID | ProteinGI |␣
↪NucleotideGI | TaxonomyID | MIMID | GeneID | ProbeID | PatentID}
<output> = XML | ASNT | ASNB | JSON | JSONP [ ?callback=<callback␣
↪name> ] | SDF | CSV | PNG | TXT
```

Contribute

The Issue Tracker (https://github.com/mcs07/PubChemPy/issues) is the best place to post any feature ideas, requests and bug reports.

If you are able to contribute changes yourself, just fork the source code (https://github.com/mcs07/PubChemPy) on GitHub, make changes and file a pull request.

Contributors

- mcs07 (https://github.com/mcs07) (Matt Swain)
- ekaakurniawan (https://github.com/ekaakurniawan) (Eka A. Kurniawan)
- zachcp (https://github.com/zachcp) (Zach Powers)
- hsiaoyi0504 (https://github.com/hsiaoyi0504) (Hsiao Yi)
- llazzaro (https://github.com/llazzaro) (Leonardo Lazzaro)
- bjodah (https://github.com/bjodah) (Björn Dahlgren)
- RickardSjogren (https://github.com/RickardSjogren) (Rickard Sjögren)

CHAPTER 4

API documentation

Comprehensive API documentation with information on every function, class and method.

API documentation

This part of the documentation is automatically generated from the PubChemPy source code and comments.

Search functions

pubchempy.**get_compounds**(*identifier*, *namespace=u'cid'*, *searchtype=None*, *as_dataframe=False*, ***kwargs*)

Retrieve the specified compound records from PubChem.

Parameters

- **identifier** – The compound identifier to use as a search query.
- **namespace** – (optional) The identifier type, one of cid, name, smiles, sdf, inchi, inchikey or formula.
- **searchtype** – (optional) The advanced search type, one of substructure, superstructure or similarity.
- **as_dataframe** – (optional) Automatically extract the *Compound* (page 22) properties into a pandas DataFrame (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) and return that.

pubchempy.**get_substances**(*identifier*, *namespace=u'sid'*, *as_dataframe=False*, ***kwargs*)

Retrieve the specified substance records from PubChem.

Parameters

- **identifier** – The substance identifier to use as a search query.
- **namespace** – (optional) The identifier type, one of sid, name or sourceid/<source name>.
- **as_dataframe** – (optional) Automatically extract the *Substance* (page 27) properties into a pandas DataFrame (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) and return that.

pubchempy.**get_assays**(*identifier*, *namespace=u'aid'*, ***kwargs*)

Retrieve the specified assay records from PubChem.

Parameters

- **identifier** – The assay identifier to use as a search query.
- **namespace** – (optional) The identifier type.

pubchempy.**get_properties**(*properties*, *identifier*, *namespace=u'cid'*, *searchtype=None*, *as_dataframe=False*, ***kwargs*)

Retrieve the specified properties from PubChem.

Parameters

- **identifier** – The compound, substance or assay identifier to use as a search query.
- **namespace** – (optional) The identifier type.
- **searchtype** – (optional) The advanced search type, one of substructure, superstructure or similarity.
- **as_dataframe** – (optional) Automatically extract the properties into a pandas DataFrame (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame).

Compound

class pubchempy.**Compound**(*record*)

Corresponds to a single record from the PubChem Compound database.

The PubChem Compound database is constructed from the Substance database using a standardization and deduplication process. Each Compound is uniquely identified by a CID.

Initialize with a record dict from the PubChem PUG REST service.

For most users, the `from_cid()` class method is probably a better way of creating Compounds.

Parameters **record** (*dict* (https://docs.python.org/2/library/stdtypes.html#dict)) – A compound record returned by the PubChem PUG REST service.

record
The raw compound record returned by the PubChem PUG REST service.

classmethod from_cid(*cid*, ***kwargs*)
Retrieve the Compound record for the specified CID.

Usage:

```
c = Compound.from_cid(6819)
```

Parameters **cid** (*int* (https://docs.python.org/2/library/functions.html#int)) – The PubChem Compound Identifier (CID).

to_dict(*properties=None*)
Return a dictionary containing Compound data. Optionally specify a list of the desired properties.

synonyms, aids and sids are not included unless explicitly specified using the properties parameter. This is because they each require an extra request.

to_series(*properties=None*)
Return a pandas `Series` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html#pandas.Series) containing Compound data. Optionally specify a list of the desired properties.

synonyms, aids and sids are not included unless explicitly specified using the properties parameter. This is because they each require an extra request.

cid
The PubChem Compound Identifier (CID).

Note: When searching using a SMILES or InChI query that is not present in the PubChem Compound database, an automatically generated record may be returned that contains properties that have been calculated on the fly. These records will not have a CID property.

---

elements
List of element symbols for atoms in this Compound.

atoms
List of *Atoms* (page 25) in this Compound.

bonds
List of *Bonds* (page 26) between *Atoms* (page 25) in this Compound.

synonyms
A ranked list of all the names associated with this Compound.

Requires an extra request. Result is cached.

sids
Requires an extra request. Result is cached.

aids
Requires an extra request. Result is cached.

charge
Formal charge on this Compound.

molecular_formula
Molecular formula.

molecular_weight
Molecular Weight.

canonical_smiles
Canonical SMILES, with no stereochemistry information.

isomeric_smiles
Isomeric SMILES.

inchi
InChI string.

inchikey
InChIKey.

iupac_name
Preferred IUPAC name.

xlogp
XLogP.

exact_mass
Exact mass.

monoisotopic_mass
Monoisotopic mass.

tpsa
Topological Polar Surface Area.

complexity
Complexity.

h_bond_donor_count
Hydrogen bond donor count.

h_bond_acceptor_count
Hydrogen bond acceptor count.

rotatable_bond_count
Rotatable bond count.

fingerprint
Raw padded and hex-encoded fingerprint, as returned by the PUG REST API.

cactvs_fingerprint
PubChem CACTVS fingerprint.

Each bit in the fingerprint represents the presence or absence of one of 881 chemical substructures.

More information at ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.txt

heavy_atom_count
Heavy atom count.

isotope_atom_count
Isotope atom count.

atom_stereo_count
Atom stereocenter count.

defined_atom_stereo_count
Defined atom stereocenter count.

undefined_atom_stereo_count
Undefined atom stereocenter count.

bond_stereo_count
Bond stereocenter count.

defined_bond_stereo_count
Defined bond stereocenter count.

undefined_bond_stereo_count
Undefined bond stereocenter count.

covalent_unit_count
Covalently-bonded unit count.

Atom

class pubchempy.**Atom**(*aid*, *number*, *x=None*, *y=None*, *z=None*, *charge=0*)
Class to represent an atom in a *Compound* (page 22).

Initialize with an atom ID, atomic number, coordinates and optional change.

Parameters

- **aid** (*int* (https://docs.python.org/2/library/functions.html#int)) – Atom ID
- **number** (*int* (https://docs.python.org/2/library/functions.html#int)) – Atomic number

- **x** (*float* (https://docs.python.org/2/library/functions.html#float)) – X coordinate.
- **y** (*float* (https://docs.python.org/2/library/functions.html#float)) – Y coordinate.
- **z** (*float* (https://docs.python.org/2/library/functions.html#float)) – (optional) Z coordinate.
- **charge** (*int* (https://docs.python.org/2/library/functions.html#int)) – (optional) Formal charge on atom.

aid = None
The atom ID within the owning Compound.

number = None
The atomic number for this atom.

x = None
The x coordinate for this atom.

y = None
The y coordinate for this atom.

z = None
The z coordinate for this atom. Will be `None` in 2D Compound records.

charge = None
The formal charge on this atom.

element
The element symbol for this atom.

to_dict()
Return a dictionary containing Atom data.

set_coordinates(*x*, *y*, *z=None*)
Set all coordinate dimensions at once.

coordinate_type
Whether this atom has 2D or 3D coordinates.

Bond

class pubchempy.**Bond**(*aid1*, *aid2*, *order=1*, *style=None*)
Class to represent a bond between two atoms in a *Compound* (page 22).

Initialize with begin and end atom IDs, bond order and bond style.

Parameters

- **aid1** (*int* (https://docs.python.org/2/library/functions.html#int)) – Begin atom ID.
- **aid2** (*int* (https://docs.python.org/2/library/functions.html#int)) – End atom ID.

- **order** (*int* (https://docs.python.org/2/library/functions.html#int)) – Bond order.

aid1 = None
ID of the begin atom of this bond.

aid2 = None
ID of the end atom of this bond.

order = None
Bond order.

style = None
Bond style annotation.

to_dict()
Return a dictionary containing Bond data.

Substance

class pubchempy.**Substance**(*record*)
Corresponds to a single record from the PubChem Substance database.

The PubChem Substance database contains all chemical records deposited in PubChem in their most raw form, before any significant processing is applied. As a result, it contains duplicates, mixtures, and some records that don't make chemical sense. This means that Substance records contain fewer calculated properties, however they do have additional information about the original source that deposited the record.

The PubChem Compound database is constructed from the Substance database using a standardization and deduplication process. Hence each Compound may be derived from a number of different Substances.

classmethod from_sid(*sid*)
Retrieve the Substance record for the specified SID.

Parameters **sid** (*int* (https://docs.python.org/2/library/functions.html#int)) – The PubChem Substance Identifier (SID).

record = None
A dictionary containing the full Substance record that all other properties are obtained from.

to_dict(*properties=None*)
Return a dictionary containing Substance data.

If the properties parameter is not specified, everything except cids and aids is included. This is because the aids and cids properties each require an extra request to retrieve.

Parameters **properties** – (optional) A list of the desired properties.

to_series(*properties=None*)

Return a pandas `Series` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html#pandas.Series) containing Substance data.

If the properties parameter is not specified, everything except cids and aids is included. This is because the aids and cids properties each require an extra request to retrieve.

Parameters **properties** – (optional) A list of the desired properties.

sid

The PubChem Substance Idenfitier (SID).

synonyms

A ranked list of all the names associated with this Substance.

source_name

The name of the PubChem depositor that was the source of this Substance.

source_id

Unique ID for this Substance within those from the same PubChem depositor source.

standardized_cid

The CID of the Compound that was produced when this Substance was standardized.

May not exist if this Substance was not standardizable.

standardized_compound

Return the *Compound* (page 22) that was produced when this Substance was standardized.

Requires an extra request. Result is cached.

deposited_compound

Return a *Compound* (page 22) produced from the unstandardized Substance record as deposited.

The resulting *Compound* (page 22) will not have a `cid` and will be missing most properties.

cids

A list of all CIDs for Compounds that were produced when this Substance was standardized.

Requires an extra request. Result is cached.

aids

A list of all AIDs for Assays associated with this Substance.

Requires an extra request. Result is cached.

Assay

class pubchempy.**Assay**(*record*)

classmethod from_aid(*aid*)
Retrieve the Assay record for the specified AID.

Parameters **aid** (*int* (https://docs.python.org/2/library/functions.html#int)) – The PubChem Assay Identifier (AID).

record = None
A dictionary containing the full Assay record that all other properties are obtained from.

to_dict(*properties=None*)
Return a dictionary containing Assay data.

If the properties parameter is not specified, everything is included.

Parameters **properties** – (optional) A list of the desired properties.

aid
The PubChem Substance Idenfitier (SID).

name
The short assay name, used for display purposes.

description
Description

project_category
A category to distinguish projects funded through MLSCN, MLPCN or from literature.

Possible values include mlscn, mlpcn, mlscn-ap, mlpcn-ap, literature-extracted, literature-author, literature-publisher, rnaigi.

comments
Comments and additional information.

results
A list of dictionaries containing details of the results from this Assay.

target
A list of dictionaries containing details of the Assay targets.

revision
Revision identifier for textual description.

aid_version
Incremented when the original depositor updates the record.

pandas functions

Each of the search functions, *get_compounds()* (page 21), *get_substances()* (page 21) and *get_properties()* (page 22) has an `as_dataframe` parameter. When set to `True`, these functions automatically extract properties from each result in the list into a pandas `DataFrame` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) and return that instead of the results themselves.

If you already have a list of Compounds or Substances, the functions below allow a `DataFrame` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) to be constructed easily.

pubchempy.**compounds_to_frame**(*compounds*, *properties=None*)

Construct a pandas `DataFrame` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) from a list of *Compound* (page 22) objects.

Optionally specify a list of the desired *Compound* (page 22) properties.

pubchempy.**substances_to_frame**(*substances*, *properties=None*)

Construct a pandas `DataFrame` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame) from a list of *Substance* (page 27) objects.

Optionally specify a list of the desired *Substance* (page 27) properties.

Exceptions

exception pubchempy.**PubChemPyError**

Base class for all PubChemPy exceptions.

exception pubchempy.**ResponseParseError**

PubChem response is uninterpretable.

exception pubchempy.**PubChemHTTPError**

Generic error class to handle all HTTP error codes.

exception pubchempy.**BadRequestError**

Request is improperly formed (syntax error in the URL, POST body, etc.).

exception pubchempy.**NotFoundError**

The input record was not found (e.g. invalid CID).

exception pubchempy.**MethodNotAllowedError**

Request not allowed (such as invalid MIME type in the HTTP Accept header).

exception pubchempy.**TimeoutError**

The request timed out, from server overload or too broad a request.

See *Avoiding TimeoutError* (page 16) for more information.

exception pubchempy.**UnimplementedError**
The requested operation has not (yet) been implemented by the server.

exception pubchempy.**ServerError**
Some problem on the server side (such as a database server down, etc.).

Changes

- As of v1.0.3, the `atoms` and `bonds` properties on *Compounds* (page 22) now return lists of *Atom* (page 25) and *Bond* (page 26) objects, rather than dicts.
- As of v1.0.2, search functions now return an empty list instead of raising a *NotFoundError* (page 30) exception when no results are found. *NotFoundError* (page 30) is still raised when attempting to create a *Compound* (page 22) using the `from_cid` class method with an invalid CID.

Index

H

I

M

N

O

P

R

S

T

U

X

Y

Z