ChemSpiPy Documentation

Release 2.0.0

Matt Swain

Sep 09, 2018

Contents

ChemSpiPy provides a way to interact with ChemSpider in Python. It allows chemical searches, chemical file downloads, depiction and retrieval of chemical properties. Here's a quick peek:

```
>>> from chemspipy import ChemSpider
>>> cs = ChemSpider('<YOUR-API-KEY>')
>>> c1 = cs.get_compound(236)  # Specify compound by ChemSpider ID
>>> c2 = cs.search('benzene')  # Search using name, SMILES, InChI, InChIKey, etc.
```

CHAPTER 1

Features

- Search compounds by synonym, SMILES, InChI, InChIKey, formula and mass.
- Get identifiers and calculated properties for any compound record in ChemSpider.
- Download compound records as a MOL file with 2D or 3D coordinates.
- Get a 2D compound depiction as a PNG image.
- Complete interface to every endpoint of the ChemSpider Web APIs.
- Supports Python versions 2.7 and 3.5+.

CHAPTER 2

User Guide

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

2.1 Introduction

ChemSpiPy is a Python wrapper that allows simple access to the web APIs offered by ChemSpider. The aim is to provide an interface for users to access and query the ChemSpider database using Python, facilitating programs that can automatically carry out the tasks that you might otherwise perform manually via the ChemSpider website.

The RSC website has full documentation for the ChemSpider APIs. It can be useful to browse through this documentation before getting started with ChemSpiPy to get an idea of what sort of features are available.

2.1.1 Obtaining an API Key

The Royal Society of Chemistry web services are currently available as an Open Developer Preview. During the preview you can make 1000 calls per month. For an increased allowance, contact api@rsc.org.

All operations require an API key. To obtain one, Register for a RSC Developers account and then Add a new key.

2.2 Installation

ChemSpiPy supports Python versions 2.7 and 3.5+.

There are two required dependencies: six and requests.

2.2.1 Option 1: Use conda (recommended)

The easiest and recommended way to install is using conda. Anaconda Python is a self-contained Python environment that is particularly useful for scientific applications. If you don't already have it, start by installing Miniconda, which includes a complete Python distribution and the conda package manager. Choose the Python 3 version, unless you have a particular reason why you must use Python 2.

To install ChemSpiPy, at the command line, run:

```
conda config --add channels conda-forge
conda install chemspipy
```

This will add the conda-forge channel to your conda config, then install ChemSpiPy and all its dependencies into your conda environment.

2.2.2 Option 2: Use pip

An alternative method is to install using pip:

```
pip install chemspipy
```

This will download the latest version of ChemSpiPy, and place it in your *site-packages* folder so it is automatically available to all your python scripts. It should also ensure that the dependencies six and requests are installed.

If you don't already have pip installed, you can install it using get-pip.py:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py
```

2.2.3 Option 3: Download the Latest Release

Alternatively, download the latest release manually and install yourself:

```
tar -xzvf ChemSpiPy-2.0.0.tar.gz
cd ChemSpiPy-2.0.0
python setup.py install
```

The setup.py command will install ChemSpiPy in your *site-packages* folder so it is automatically available to all your python scripts.

2.2.4 Option 4: Clone the Repository

The latest development version of ChemSpiPy is always available on GitHub. 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/ChemSpiPy.git
cd ChemSpiPy
python setup.py install
```

2.3 Getting Started

This page gives a introduction on how to get started with ChemSpiPy.

2.3.1 Before We Start

- Make sure you have *installed ChemSpiPy*.
- *Obtain an API key* from the ChemSpider web site.

2.3.2 First Steps

Start by importing ChemSpider:

```
>>> from chemspipy import ChemSpider
```

Then connect to ChemSpider by creating a `ChemSpider` instance using your API key:

```
>>> cs = ChemSpider('<YOUR-API-KEY>')
```

All your interaction with the ChemSpider database should now happen through this ChemSpider object, `cs`.

2.3.3 Retrieve a Compound

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

Let's get the Compound with ChemSpider ID 2157:

```
>>> c = cs.get_compound(2157)
```

Now we have a *`Compound`* object called `c`. We can get various identifiers and calculated properties from this object:

```
>>> print(c.molecular_formula)
C_{9}H_{8}O_{4}
>>> print(c.molecular_weight)
180.1574
>>> print(c.smiles)
CC(=O)Oc1ccccc1C(=O)O
>>> print(c.common_name)
Aspirin
```

2.3.4 Search for a Name

What if you don't know the ChemSpider ID of the Compound you want? Instead use the `search` method:

```
>>> for result in cs.search('Glucose'):
...     print(result)
Compound(5589)
Compound(58238)
Compound(71358)
Compound(96749)
Compound(2006622)
Compound(5341883)
Compound(5360239)
Compound(9129332)
Compound(9281077)
Compound(9312824)
Compound(9484839)
Compound(9655623)
```

The `search` method accepts any identifer that ChemSpider can interpret, including names, registry numbers, SMILES and InChI.

That's a quick taster of the basic ChemSpiPy functionality. Read on for more some more advanced usage examples.

2.4 Compound

Many ChemSpiPy search methods return *`Compound`* objects, which provide more functionality that a simple list of ChemSpider IDs. The primary benefit is allowing easy access to further compound properties after performing a search.

2.4.1 Creating a Compound

The easiest way to create a *`Compound`* for a given ChemSpider ID is to use the *`get_compound()`* method:

```
>>> compound = cs.get_compound(2157)
```

Alternatively, a *`Compound`* can be instantiated directly:

```
>>> compound = Compound(cs, 2157)
```

Either way, no requests are made to the ChemSpider servers until specific *`Compound`* properties are requested:

```
>>> print(compound.molecular_formula)
C_{9}H_{8}O_{4}
>>> print(compound.molecular_weight)
180.1574
>>> print(compound.smiles)
CC(=O)Oc1ccccc1C(=O)O
>>> print(compound.common_name)
Aspirin
```

Properties are cached locally after the first time they are retrieved, speeding up subsequent access and reducing the number of unnecessary requests to the ChemSpider servers.

2.4.2 External References

Get a list of all external references for a given compound using the *`external_references`* property:

```
>>> refs = compound.external_references
>>> print(len(refs))
28181
>>> print(refs[0])
{'source': 'ChemBank', 'sourceUrl': 'http://chembank.broadinstitute.org/', 'externalId
↪': 'DivK1c_000555', 'externalUrl': 'http://chembank.broad.harvard.edu/chemistry/
↪viewMolecule.htm?cbid=1171'}
```

Each reference is a dict with details for an external source. The list of references can be very large and slow to retrieve for popular compounds, so it is possible to filter it by datasource. To do this, use the *`get_external_references()`* method directly:

```
>>> refs = cs.get_external_references(2157, datasources=['PubChem'])
>>> print(refs)
[{'source': 'PubChem', 'sourceUrl': 'http://pubchem.ncbi.nlm.nih.gov/', 'externalId':␣
↪2244, 'externalUrl': 'http://pubchem.ncbi.nlm.nih.gov/summary/summary.cgi?cid=2244'}
↪]
```

See the *Data Sources* documentation for how to get a list of all available data sources.

2.4.3 Searching for Compounds

See the *searching documentation* for full details.

2.4.4 Implementation Details

Each `Compound` object is a simple wrapper around a ChemSpider ID. Behind the scenes, the property methods use the `get_details()`, `convert()`, `get_image()`, and `get_external_references()` API methods to retrieve the relevant information. It is possible to use these API methods directly if required:

```
>>> info = cs.get_details(2157)
>>> print(info.keys())
dict_keys(['id', 'smiles', 'formula', 'averageMass', 'molecularWeight',
↪'monoisotopicMass', 'nominalMass', 'commonName', 'referenceCount', 'dataSourceCount
↪', 'pubMedCount', 'rscCount', 'mol2D', 'mol3D'])
>>> print(info['smiles'])
CC(=O)Oc1ccccc1C(=O)O
```

Results are returned as a python dictionary that is derived directly from the ChemSpider API JSON response.

2.4.5 Compound Properties

class chemspipy.objects.**Compound**

Compound.**record_id**
Compound record ID.

Return type int

Compound.**image_url**
Return the URL of a PNG image of the 2D chemical structure.

Return type string

Compound.**molecular_formula**
Return the molecular formula for this Compound.

Return type string

Compound.**inchi**
Return the InChI for this Compound.

Return type string

Compound.**inchikey**
Return the InChIKey for this Compound.

Return type string

`Compound.`**`average_mass`**
Return the average mass of this Compound.

Return type float

`Compound.`**`molecular_weight`**
Return the molecular weight of this Compound.

Return type float

`Compound.`**`monoisotopic_mass`**
Return the monoisotopic mass of this Compound.

Return type float

`Compound.`**`nominal_mass`**
Return the nominal mass of this Compound.

Return type float

`Compound.`**`common_name`**
Return the common name for this Compound.

Return type string

`Compound.`**`mol_2d`**
Return the MOL file for this Compound with 2D coordinates.

Return type string

`Compound.`**`mol_3d`**
Return the MOL file for this Compound with 3D coordinates.

Return type string

`Compound.`**`image`**
Return a 2D depiction of this Compound.

Return type bytes

`Compound.`**`external_references`**
Return external references for this Compound.

Return type list[dict]

2.5 Searching

ChemSpiPy provides a number of different ways to search ChemSpider.

2.5.1 Compound Search

The main ChemSpiPy search method functions in a similar way to the main search box on the ChemSpider website. Just provide any type of query, and ChemSpider will interpret it and provide the most relevant results:

```
>>> cs.search('O=C(OCC)C')
Results([Compound(8525)])
>>> cs.search('glucose')
Results([Compound(5589), Compound(58238), Compound(71358), Compound(96749),␣
↪Compound(2006622), Compound(5341883), Compound(5360239), Compound(9129332),␣
↪Compound(9281077), Compound(9312824), Compound(9484839), Compound(9655623)])
```

(continues on next page)

(continued from previous page)

```
>>> cs.search('2157')
Results([Compound(2157)])
```

The supported query types include systematic names, synonyms, trade names, registry numbers, molecular formula, SMILES, InChI and InChIKey.

The `Results` object that is returned can be treated just like any regular python list. For example, you can iterate over the results:

```
>>> for result in cs.search('Glucose'):
...     print(result.record_id)
5589
58238
71358
96749
2006622
5341883
5360239
9129332
9281077
9312824
9484839
9655623
```

The `Results` object also provides the time taken to perform the search, and a message that explains how the query type was resolved:

```
>>> r = cs.search('Glucose')
>>> print(r.duration)
0:00:00.513406
>>> print(r.message)
Found by approved synonym
```

2.5.2 Asynchronous Searching

Certain types of search can sometimes take slightly longer, which can be inconvenient if the search method blocks the Python interpreter until the search results are returned. Fortunately, the ChemSpiPy search method works asynchronously.

Once a search is executed, ChemSpiPy immediately returns the `Results` object, which is actually empty at first:

```
>>> results = cs.search('O=C(OCC)C')
>>> print(results.ready())
False
```

In a background thread, ChemSpiPy is making the search request and waiting for the response. But in the meantime, it is possible to continue performing other tasks in the main Python interpreter process. Call `ready()` at any point to check if the results have been returned and are available.

Any attempt to access the results will just block until the results are ready, like a simple synchronous search. To manually block the main thread until the results are ready, use the `wait()` method:

```
>>> results.wait()
>>> results.ready()
True
```

For more detailed information about the status of a search, use the *status* property:

```
>>> results.status
'Created'
>>> results.wait()
>>> results.status
'Complete'
```

The possible statuses are `Created`, `Failed`, `Unknown`, `Suspended`, `Complete`.

2.5.3 Search by Formula

Searching by molecular formula is supported by the main *search()* method, but there is the possibility that a formula could be interpreted as a name or SMILES or another query type. To specifically search by formula, use:

```
>>> cs.search_by_formula('C44H30N4Zn')
[Compound(436642), Compound(3232330), Compound(24746832), Compound(26995124)]
```

2.5.4 Search by Mass

It is also possible to search ChemSpider by mass by specifying a certain range:

```
>>> cs.search_by_mass(680, 0.001)
[Compound(8298180), Compound(12931939), Compound(12931969), Compound(21182158)]
```

The first parameter specifies the desired molecular mass, while the second parameter specifies the allowed $\pm$ range of values.

2.6 Miscellaneous

2.6.1 Data Sources

Get a list of data sources in ChemSpider using the *get_datasources()* method:

```
>>> cs.get_datasources()
['Abacipharm', 'Abblis Chemicals', 'Abcam', 'ABI Chemicals', 'Abmole Bioscience',
↪'ACB Blocks', 'Accela ChemBio', ... ]
```

2.6.2 Format Conversion

Convert between different molecular representations using the *convert()* method:

```
>>> cs.convert('c1ccccc1', 'SMILES', 'InChI')
'InChI=1S/C6H6/c1-2-4-6-5-3-1/h1-6H'
```

Allowed conversions:

- From `InChI` to `InChIKey`
- From `InChI` to `Mol`
- From `InChI` to `SMILES`

- From `InChIKey` to `InChI`
- From `InChIKey` to `Mol`
- From `Mol` to `InChI`
- From `Mol` to `InChIKey`
- From `SMILES` to `InChI`

2.7 Advanced

2.7.1 Keep Your API Key Secret

Be careful not to include your API key when sharing code. A simple way to ensure this doesn't happen by accident is to store your API key as an environment variable that can be specified in your *.bash_profile* or *.zshrc* file:

```
export CHEMSPIDER_API_KEY=<YOUR-API-KEY>
```

This can then be retrieved in your scripts using `os.environ`:

```
>>> api_key = os.environ['CHEMSPIDER_API_KEY']
>>> cs = ChemSpider(api_key)
```

2.7.2 Specify a User Agent

As well as using your API key, it is possible to identify your program to the ChemSpider servers using a User Agent string.

You can specify a custom User Agent through ChemSpiPy through the optional `user_agent` parameter to the ChemSpider class:

```
>>> from chemspipy import ChemSpider
>>> cs = ChemSpider('<YOUR-API-KEY>', user_agent='My program 1.3, ChemSpiPy 2.0.0,␣
↪Python 3.6')
```

2.7.3 Logging

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

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

The logger is named 'chemspipy'. There is more information on logging in the Python logging documentation.

CHAPTER 3

API Documentation

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

3.1 API Documentation

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

3.1.1 chemspipy.api

Core API for interacting with ChemSpider web services.

class chemspipy.api.**ChemSpider**

Provides access to the ChemSpider API.

Usage:

```
>>> from chemspipy import ChemSpider
>>> cs = ChemSpider('<YOUR-API-KEY>')
```

Parameters

- **api_key** (*string*) – Your ChemSpider API key.
- **user_agent** (*string*) – (Optional) Identify your application to ChemSpider servers.
- **api_url** (*string*) – (Optional) API server. Default https://api.rsc.org.
- **api_version** (*string*) – (Optional) API version. Default v1.

request (*method*, *api*, *namespace*, *endpoint*, *params=None*, *json=None*)

Make a request to the ChemSpider API.

Parameters

- **method** (*string*) – HTTP method.

- **api** (*string*) – Top-level API, e.g. compounds.
- **namespace** (*string*) – API namespace, e.g. filter, lookups, records, or tools.
- **endpoint** (*string*) – Web service endpoint URL.
- **params** (*dict*) – Query parameters to add to the URL.
- **json** (*dict*) – JSON data to send in the request body.

Returns Web Service response JSON.

Return type dict

get (*api*, *namespace*, *endpoint*, *params=None*)
Convenience method for making GET requests.

Parameters

- **api** (*string*) – Top-level API, e.g. compounds.
- **namespace** (*string*) – API namespace, e.g. filter, lookups, records, or tools.
- **endpoint** (*string*) – Web service endpoint URL.
- **params** (*dict*) – Query parameters to add to the URL.

Returns Web Service response JSON.

Return type dict

post (*api*, *namespace*, *endpoint*, *json=None*)
Convenience method for making POST requests.

Parameters

- **api** (*string*) – Top-level API, e.g. compounds.
- **namespace** (*string*) – API namespace, e.g. filter, lookups, records, or tools.
- **endpoint** (*string*) – Web service endpoint URL.
- **json** (*dict*) – JSON data to send in the request body.

Returns Web Service response content.

Return type dict or string

get_compound (*csid*)
Return a Compound object for a given ChemSpider ID.

Parameters **csid** (*string|int*) – ChemSpider ID.

Returns The Compound with the specified ChemSpider ID.

Return type *Compound*

get_compounds (*csids*)
Return a list of Compound objects, given a list ChemSpider IDs.

Parameters **csids** (*list[string|int]*) – List of ChemSpider IDs.

Returns List of Compounds with the specified ChemSpider IDs.

Return type list[*Compound*]

search (*query*, *order=None*, *direction='ascending'*, *raise_errors=False*)
Search ChemSpider for the specified query and return the results.

The accepted values for `order` are: *`RECORD_ID`*, *`MASS_DEFECT`*, *`MOLECULAR_WEIGHT`*, *`REFERENCE_COUNT`*, *`DATASOURCE_COUNT`*, *`PUBMED_COUNT`* or *`RSC_COUNT`*.

Parameters

- **`query`** (*`string|int`*) – Search query.
- **`order`** (*`string`*) – (Optional) Field to sort the result by.
- **`direction`** (*`string`*) – (Optional) *`ASCENDING`* or *`DESCENDING`*.
- **`raise_errors`** (*`bool`*) – (Optional) If True, raise exceptions. If False, store on Results `exception` property.

Returns Search Results list.

Return type *`Results`*

`get_datasources``()`

Get the list of datasources in ChemSpider.

Many other endpoints let you restrict which sources are used to lookup the requested query. Restricting the sources makes queries faster.

Returns List of datasources.

Return type list[string]

`get_details``(`*record_id, fields=['SMILES', 'Formula', 'AverageMass', 'MolecularWeight', 'MonoisotopicMass', 'NominalMass', 'CommonName', 'ReferenceCount', 'DataSourceCount', 'PubMedCount', 'RSCCount', 'Mol2D', 'Mol3D']*`)`

Get details for a compound record.

The available fields are listed in *`FIELDS`*.

Parameters

- **`record_id`** (*`int`*) – Record ID.
- **`fields`** (*`list[string]`*) – (Optional) List of fields to include in the result.

Returns Record details.

Return type dict

`get_details_batch``(`*record_ids, fields=['SMILES', 'Formula', 'AverageMass', 'MolecularWeight', 'MonoisotopicMass', 'NominalMass', 'CommonName', 'ReferenceCount', 'DataSourceCount', 'PubMedCount', 'RSCCount', 'Mol2D', 'Mol3D']*`)`

Get details for a list of compound records.

The available fields are listed in *`FIELDS`*.

Parameters

- **`record_ids`** (*`list[int]`*) – List of record IDs (up to 100).
- **`fields`** (*`list[string]`*) – (Optional) List of fields to include in the results.

Returns List of record details.

Return type list[dict]

`get_external_references``(`*record_id, datasources=None*`)`

Get external references for a compound record.

Optionally filter the results by data source. Use *`get_datasources()`* to get the available datasources.

Parameters

- **record_id** (*int*) – Record ID.
- **datasources** (*list[string]*) – (Optional) List of datasources to restrict the results to.

Returns External references.

Return type list[dict]

get_image (*record_id*)
Get image for a compound record.

Parameters **record_id** (*int*) – Record ID.

Returns Image.

Return type bytes

get_mol (*record_id*)
Get MOLfile for a compound record.

Parameters **record_id** (*int*) – Record ID.

Returns MOLfile.

Return type string

filter_element (*include_elements*, *exclude_elements=None*, *include_all=False*, *complexity=None*, *isotopic=None*, *order=None*, *direction=None*)
Search compounds by element.

Set include_all to true to only consider records that contain all of the elements in `include_elements`, otherwise all records that contain any of the elements will be returned.

A compound with a complexity of 'multiple' has more than one disconnected system in it or a metal atom or ion.

The accepted values for `order` are: *RECORD_ID*, *MASS_DEFECT*, *MOLECULAR_WEIGHT*, *REFERENCE_COUNT*, *DATASOURCE_COUNT*, *PUBMED_COUNT* or *RSC_COUNT*.

Parameters

- **include_elements** (*list[string]*) – List of up to 15 elements to search for compounds containing.
- **exclude_elements** (*list[string]*) – List of up to 100 elements to exclude compounds containing.
- **include_all** (*bool*) – (Optional) Whether to only include compounds that have all include_elements.
- **complexity** (*string*) – (Optional) 'any', 'single', or 'multiple'
- **isotopic** (*string*) – (Optional) 'any', 'labeled', or 'unlabeled'.
- **order** (*string*) – (Optional) Field to sort the result by.
- **direction** (*string*) – (Optional) *ASCENDING* or *DESCENDING*.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

filter_formula (*formula*, *datasources=None*, *order=None*, *direction=None*)
Search compounds by formula.

Optionally filter the results by data source. Use *get_datasources()* to get the available datasources.

The accepted values for `order` are: *RECORD_ID*, *MASS_DEFECT*, *MOLECULAR_WEIGHT*, *REFERENCE_COUNT*, *DATASOURCE_COUNT*, *PUBMED_COUNT* or *RSC_COUNT*.

Parameters

- **formula** (*string*) – Molecular formula.
- **datasources** (*list[string]*) – (Optional) List of datasources to restrict the results to.
- **order** (*string*) – (Optional) Field to sort the result by.
- **direction** (*string*) – (Optional) *ASCENDING* or *DESCENDING*.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

filter_formula_batch (*formulas*, *datasources=None*, *order=None*, *direction=None*)
Search compounds with a list of formulas.

Optionally filter the results by data source. Use `get_datasources()` to get the available datasources.

The accepted values for `order` are: *RECORD_ID*, *MASS_DEFECT*, *MOLECULAR_WEIGHT*, *REFERENCE_COUNT*, *DATASOURCE_COUNT*, *PUBMED_COUNT* or *RSC_COUNT*.

Parameters

- **formulas** (*list[string]*) – Molecular formula.
- **datasources** (*list[string]*) – (Optional) List of datasources to restrict the results to.
- **order** (*string*) – (Optional) Field to sort the result by.
- **direction** (*string*) – (Optional) *ASCENDING* or *DESCENDING*.

Returns Query ID that may be passed to `filter_formula_batch_status` and `filter_formula_batch_results`.

Return type string

filter_formula_batch_status (*query_id*)
Get formula batch filter status using a query ID that was returned by a previous filter request.

Parameters **query_id** (*string*) – Query ID from a previous formula batch filter request.

Returns Status dict with 'status', 'count', and 'message' fields.

Return type dict

filter_formula_batch_results (*query_id*)
Get formula batch filter results using a query ID that was returned by a previous filter request.

Each result is a dict containing a `formula` key and a `results` key.

Parameters **query_id** (*string*) – Query ID from a previous formula batch filter request.

Returns List of results.

Return type list[dict]

filter_inchi (*inchi*)
Search compounds by InChI.

Parameters **inchi** (*string*) – InChI.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

`filter_inchikey`(*inchikey*)

Search compounds by InChIKey.

Parameters **`inchikey`** (*`string`*) – InChIKey.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

`filter_intrinsicproperty`(*formula=None*, *molecular_weight=None*, *nominal_mass=None*, *average_mass=None*, *monoisotopic_mass=None*, *molecular_weight_range=None*, *nominal_mass_range=None*, *average_mass_range=None*, *monoisotopic_mass_range=None*, *complexity=None*, *isotopic=None*, *order=None*, *direction=None*)

Search compounds by intrinsic property, such as formula and mass.

At least one of formula, molecular_weight, nominal_mass, average_mass, monoisotopic_mass must be specified.

A compound with a complexity of 'multiple' has more than one disconnected system in it or a metal atom or ion.

The accepted values for `order` are: *`RECORD_ID`*, *`MASS_DEFECT`*, *`MOLECULAR_WEIGHT`*, *`REFERENCE_COUNT`*, *`DATASOURCE_COUNT`*, *`PUBMED_COUNT`* or *`RSC_COUNT`*.

Parameters

- **`formula`** (*`string`*) – Molecular formula.
- **`molecular_weight`** (*`float`*) – Molecular weight.
- **`nominal_mass`** (*`float`*) – Nominal mass.
- **`average_mass`** (*`float`*) – Average mass.
- **`monoisotopic_mass`** (*`float`*) – Monoisotopic mass.
- **`molecular_weight_range`** (*`float`*) – Molecular weight range.
- **`nominal_mass_range`** (*`float`*) – Nominal mass range.
- **`average_mass_range`** (*`float`*) – Average mass range.
- **`monoisotopic_mass_range`** (*`float`*) – Monoisotopic mass range.
- **`complexity`** (*`string`*) – (Optional) 'any', 'single', or 'multiple'
- **`isotopic`** (*`string`*) – (Optional) 'any', 'labeled', or 'unlabeled'.
- **`order`** (*`string`*) – (Optional) Field to sort the result by.
- **`direction`** (*`string`*) – (Optional) *`ASCENDING`* or *`DESCENDING`*.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

`filter_mass`(*mass*, *mass_range*, *datasources=None*, *order=None*, *direction=None*)

Search compounds by mass.

Filter to compounds within `mass_range` of the given `mass`.

Optionally filter the results by data source. Use *`get_datasources()`* to get the available datasources.

The accepted values for `order` are: *`RECORD_ID`*, *`MASS_DEFECT`*, *`MOLECULAR_WEIGHT`*, *`REFERENCE_COUNT`*, *`DATASOURCE_COUNT`*, *`PUBMED_COUNT`* or *`RSC_COUNT`*.

Parameters

- **mass** (*float*) – Mass between 1 and 11000 Atomic Mass Units.
- **mass_range** (*float*) – Mass range between 0.0001 and 100 Atomic Mass Units.
- **datasources** (*list[string]*) – (Optional) List of datasources to restrict the results to.
- **order** (*string*) – (Optional) Field to sort the result by.
- **direction** (*string*) – (Optional) *ASCENDING* or *DESCENDING*.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

filter_mass_batch(*masses*, *datasources=None*, *order=None*, *direction=None*)
Search compounds with a list of masses and mass ranges.

The `masses` parameter should be a list of tuples, each with two elements: A mass, and a mass range:

```
qid = cs.filter_mass_batch(masses=[(12, 0.001), (24, 0.001)])
```

Optionally filter the results by data source. Use *get_datasources()* to get the available datasources.

The accepted values for `order` are: *RECORD_ID*, *MASS_DEFECT*, *MOLECULAR_WEIGHT*, *REFERENCE_COUNT*, *DATASOURCE_COUNT*, *PUBMED_COUNT* or *RSC_COUNT*.

Parameters

- **float]] masses** (*list[tuple[float,*) – List of (mass, range) tuples.
- **datasources** (*list[string]*) – (Optional) List of datasources to restrict the results to.
- **order** (*string*) – (Optional) Field to sort the result by.
- **direction** (*string*) – (Optional) *ASCENDING* or *DESCENDING*.

Returns Query ID that may be passed to `filter_formula_batch_status` and `filter_formula_batch_results`.

Return type string

filter_mass_batch_status(*query_id*)
Get formula batch filter status using a query ID that was returned by a previous filter request.

Parameters **query_id** (*string*) – Query ID from a previous formula batch filter request.

Returns Status dict with 'status', 'count', and 'message' fields.

Return type dict

filter_mass_batch_results(*query_id*)
Get formula batch filter results using a query ID that was returned by a previous filter request.

Each result is a dict containing a `formula` key and a `results` key.

Parameters **query_id** (*string*) – Query ID from a previous formula batch filter request.

Returns List of results.

Return type list[dict]

`filter_name`(*name*, *order=None*, *direction=None*)
Search compounds by name.

The accepted values for `order` are: *`RECORD_ID`*, *`MASS_DEFECT`*, *`MOLECULAR_WEIGHT`*, *`REFERENCE_COUNT`*, *`DATASOURCE_COUNT`*, *`PUBMED_COUNT`* or *`RSC_COUNT`*.

Parameters

- **`name`** (*`string`*) – Compound name.
- **`order`** (*`string`*) – (Optional) Field to sort the result by.
- **`direction`** (*`string`*) – (Optional) *`ASCENDING`* or *`DESCENDING`*.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

`filter_smiles`(*smiles*)
Search compounds by SMILES.

Parameters **`smiles`** (*`string`*) – Compound SMILES.

Returns Query ID that may be passed to `filter_status` and `filter_results`.

Return type string

`filter_status`(*query_id*)
Get filter status using a query ID that was returned by a previous filter request.

Parameters **`query_id`** (*`string`*) – Query ID from a previous filter request.

Returns Status dict with ‘status’, ‘count’, and ‘message’ fields.

Return type dict

`filter_results`(*query_id*, *start=None*, *count=None*)
Get filter results using a query ID that was returned by a previous filter request.

Parameters

- **`query_id`** (*`string`*) – Query ID from a previous filter request.
- **`start`** (*`int`*) – Zero-based results offset.
- **`count`** (*`int`*) – Number of results to return.

Returns List of results.

Return type list[int]

`filter_results_sdf`(*query_id*)
Get filter results as SDF file using a query ID that was returned by a previous filter request.

Parameters **`query_id`** (*`string`*) – Query ID from a previous filter request.

Returns SDF file containing the results.

Return type bytes

`convert`(*input*, *input_format*, *output_format*)
Convert a chemical from one format to another.

Format: `SMILES`, `InChI`, `InChIKey` or `Mol`.

Allowed conversions: from InChI to InChIKey, from InChI to Mol file, from InChI to SMILES, from InChIKey to InChI, from InChIKey to Mol file, from Mol file to InChI, from Mol file to InChIKey, from SMILES to InChI.

Parameters

- **`input`** (*`string`*) – Input chemical.
- **`input_format`** (*`string`*) – Input format.
- **`output_format`** (*`string`*) – Output format.

Returns Input chemical in output format.

Return type string

`validate_inchikey`(*inchikey*)
Return whether `inchikey` is valid.

Parameters **`inchikey`** (*`string`*) – The InChIKey to validate.

Returns Whether the InChIKey is valid.

Return type bool

`get_databases`()
Get the list of datasources in ChemSpider.

Deprecated since version 2.0.0: Use *`get_datasources()`* instead.

`get_extended_compound_info`(*csid*)
Get extended record details for a CSID.

Deprecated since version 2.0.0: Use *`get_details()`* instead.

Parameters **`csid`** (*`string|int`*) – ChemSpider ID.

`get_extended_compound_info_list`(*csids*)
Get extended record details for a list of CSIDs.

Deprecated since version 2.0.0: Use *`get_details_batch()`* instead.

Parameters **`csids`** (*`list[string|int]`*) – ChemSpider IDs.

`get_extended_mol_compound_info_list`(*csids*, *mol_type='2d'*, *include_reference_counts=False*, *include_external_references=False*)
Get extended record details (including MOL) for a list of CSIDs.

A maximum of 250 CSIDs can be fetched per request.

Deprecated since version 2.0.0: Use *`get_details_batch()`* instead.

Parameters

- **`csids`** (*`list[string|int]`*) – ChemSpider IDs.
- **`mol_type`** (*`string`*) – `MOL2D`, `MOL3D` or `BOTH`.
- **`include_reference_counts`** (*`bool`*) – Whether to include reference counts.
- **`include_external_references`** (*`bool`*) – Whether to include external references.

`get_record_mol`(*csid*, *calc3d=False*)
Get ChemSpider record in MOL format.

Deprecated since version 2.0.0: Use *`get_mol()`* instead.

Parameters

- **`csid`** (*`string|int`*) – ChemSpider ID.

- **calc3d** (*bool*) – Whether 3D coordinates should be calculated before returning record data.

async_simple_search(*query*)
Search ChemSpider with arbitrary query, returning results in order of the best match found.

This method returns a transaction ID which can be used with other methods to get search status and results.

Deprecated since version 2.0.0: Use `filter_name()` instead.

Parameters **query** (*string*) – Search query - a name, SMILES, InChI, InChIKey, CSID, etc.

Returns Transaction ID.

Return type string

async_simple_search_ordered(*query*, *order='csid'*, *direction='ascending'*)
Search ChemSpider with arbitrary query, returning results with a custom order.

This method returns a transaction ID which can be used with other methods to get search status and results.

Deprecated since version 2.0.0: Use `filter_name()` instead.

Parameters

- **query** (*string*) – Search query - a name, SMILES, InChI, InChIKey, CSID, etc.
- **order** (*string*) – (Optional) Field to sort the result by.
- **direction** (*string*) – (Optional) `ASCENDING` or `DESCENDING`.

Returns Transaction ID.

Return type string

get_async_search_status(*rid*)
Check the status of an asynchronous search operation.

Deprecated since version 2.0.0: Use `filter_status()` instead.

Parameters **rid** (*string*) – A transaction ID, returned by an asynchronous search method.

Returns Unknown, Created, Scheduled, Processing, Suspended, PartialResultReady, ResultReady, Failed, TooManyRecords

Return type string

get_async_search_status_and_count(*rid*)
Check the status of an asynchronous search operation. If ready, a count and message are also returned.

Deprecated since version 2.0.0: Use `filter_status()` instead.

Parameters **rid** (*string*) – A transaction ID, returned by an asynchronous search method.

Return type dict

get_async_search_result(*rid*)
Get the results from a asynchronous search operation.

Deprecated since version 2.0.0: Use `filter_results()` instead.

Parameters **rid** (*string*) – A transaction ID, returned by an asynchronous search method.

Returns A list of Compounds.

Return type list[`Compound`]

get_async_search_result_part(*rid*, *start=0*, *count=-1*)
Get a slice of the results from a asynchronous search operation.

Deprecated since version 2.0.0: Use `filter_results()` instead.

Parameters

- **rid** (*string*) – A transaction ID, returned by an asynchronous search method.
- **start** (*int*) – The number of results to skip.
- **count** (*int*) – The number of results to return. -1 returns all through to end.

Returns A list of Compounds.

Return type list[*Compound*]

get_compound_info(*csid*)
Get SMILES, StdInChI and StdInChIKey for a given CSID.

Deprecated since version 2.0.0: Use `get_details()` instead.

Parameters **csid** (*string|int*) – ChemSpider ID.

Return type dict

get_compound_thumbnail(*csid*)
Get PNG image as binary data.

Deprecated since version 2.0.0: Use `get_image()` instead.

Parameters **csid** (*string|int*) – ChemSpider ID.

Return type bytes

simple_search(*query*)
Search ChemSpider with arbitrary query.

Deprecated since version 2.0.0: Use `search()` instead.

Parameters **query** (*string*) – Search query - a chemical name.

Returns Search Results list.

Return type *Results*

chemspipy.api.**ASCENDING = 'ascending'**
Ascending sort direction

chemspipy.api.**DESCENDING = 'descending'**
Descending sort direction

chemspipy.api.**RECORD_ID = 'record_id'**
Record ID sort order

chemspipy.api.**MASS_DEFECT = 'mass_defect'**
Mass defect sort order

chemspipy.api.**MOLECULAR_WEIGHT = 'molecular_weight'**
Molecular weight sort order

chemspipy.api.**REFERENCE_COUNT = 'reference_count'**
Reference count sort order

chemspipy.api.**DATASOURCE_COUNT = 'datasource_count'**
Datasource count sort order

chemspipy.api.**PUBMED_COUNT = 'pubmed_count'**
Pubmed count sort order

chemspipy.api.**RSC_COUNT = 'rsc_count'**
RSC count sort order

chemspipy.api.**ORDERS = {'csid': 'recordId', 'datasource_count': 'dataSourceCount', 'mass_**
Map sort orders to strings required by REST API.

chemspipy.api.**DIRECTIONS = {'ascending': 'ascending', 'descending': 'descending'}**
Map sort directions to strings required by REST API.

chemspipy.api.**FIELDS = ['SMILES', 'Formula', 'AverageMass', 'MolecularWeight', 'Monoisotop**
All available compound details fields.

3.1.2 chemspipy.objects

Objects returned by ChemSpiPy API methods.

class chemspipy.objects.**Compound**(*cs*, *record_id*)
A class for retrieving and caching details about a specific ChemSpider record.

The purpose of this class is to provide access to various parts of the ChemSpider API that return information about a compound given its ChemSpider ID. Information is loaded lazily when requested, and cached for future access.

Parameters

- **cs** (ChemSpider) – ChemSpider session.
- **record_id** (*int* | *string*) – Compound record ID.

record_id
Compound record ID.

Return type int

csid
ChemSpider ID.

Deprecated since version 2.0.0: Use *record_id* instead.

Return type int

image_url
Return the URL of a PNG image of the 2D chemical structure.

Return type string

molecular_formula
Return the molecular formula for this Compound.

Return type string

smiles
Return the SMILES for this Compound.

Return type string

stdinchi
Return the Standard InChI for this Compound.

Deprecated since version 2.0.0: Use *inchi* instead.

Return type string

`stdinchikey`
Return the Standard InChIKey for this Compound.

Deprecated since version 2.0.0: Use *`inchikey`* instead.

Return type string

`inchi`
Return the InChI for this Compound.

Return type string

`inchikey`
Return the InChIKey for this Compound.

Return type string

`average_mass`
Return the average mass of this Compound.

Return type float

`molecular_weight`
Return the molecular weight of this Compound.

Return type float

`monoisotopic_mass`
Return the monoisotopic mass of this Compound.

Return type float

`nominal_mass`
Return the nominal mass of this Compound.

Return type float

`common_name`
Return the common name for this Compound.

Return type string

`mol_2d`
Return the MOL file for this Compound with 2D coordinates.

Return type string

`mol_3d`
Return the MOL file for this Compound with 3D coordinates.

Return type string

`image`
Return a 2D depiction of this Compound.

Return type bytes

`external_references`
Return external references for this Compound.

Return type list[dict]

3.1.3 chemspipy.search

A wrapper for asynchronous search requests.

class `chemspipy.search.`**`Results`**(*cs*, *searchfunc*, *searchargs*, *raise_errors=False*, *max_requests=40*)

Container class to perform a search on a background thread and hold the results when ready.

Generally shouldn't be instantiated directly. See `search()` instead.

Parameters

- **cs** (`ChemSpider`) – `ChemSpider` session.
- **searchfunc** (*function*) – Search function that returns a transaction ID.
- **searchargs** (*tuple*) – Arguments for the search function.
- **raise_errors** (*bool*) – If True, raise exceptions. If False, store on `exception` property.
- **max_requests** (*int*) – Maximum number of times to check if search results are ready.

`ready`()

Return True if the search finished.

Return type bool

`success`()

Return True if the search finished with no errors.

Return type bool

`wait`()

Block until the search has completed and optionally raise any resulting exception.

`status`

Current status string returned by ChemSpider.

Returns 'Unknown', 'Created', 'Scheduled', 'Processing', 'Suspended', 'PartialResultReady', 'ResultReady'

Return type string

`exception`

Any Exception raised during the search. Blocks until the search is finished.

`qid`

Search query ID.

Return type string

`message`

A contextual message about the search. Blocks until the search is finished.

Return type string

`count`

The number of search results. Blocks until the search is finished.

Return type int

`duration`

The time taken to perform the search. Blocks until the search is finished.

Return type `datetime.timedelta`

sdf
Get an SDF containing all the search results.

Returns SDF containing the search results.

Return type bytes

3.1.4 chemspipy.errors

Exceptions raised by ChemSpiPy.

exception `chemspipy.errors.`**`ChemSpiPyError`**
Root ChemSpiPy Exception.

exception `chemspipy.errors.`**`ChemSpiPyHTTPError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)
Base exception to handle HTTP errors.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

`MESSAGE = 'ChemSpiPy Error'`
Default message if none supplied. Override in subclasses.

exception `chemspipy.errors.`**`ChemSpiPyBadRequestError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)
Raised for a bad request.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

exception `chemspipy.errors.`**`ChemSpiPyAuthError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)
Raised when API key authorization fails.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

exception `chemspipy.errors.`**`ChemSpiPyNotFoundError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)
Raised when the requested resource was not found.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

exception `chemspipy.errors.`**`ChemSpiPyMethodError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)
Raised when an invalid HTTP method is used.

Parameters

- **`message`** (*`string|bytes`*) – Error message.

- **`http_code`** – HTTP code.

`exception` `chemspipy.errors.`**`ChemSpiPyPayloadError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)

Raised when a request payload is too large.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

`exception` `chemspipy.errors.`**`ChemSpiPyRateError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)

Raised when too many requests are sent in a given amount of time.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

`exception` `chemspipy.errors.`**`ChemSpiPyServerError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)

Raised when an internal server error occurs.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

`exception` `chemspipy.errors.`**`ChemSpiPyUnavailableError`**(*message=None*, *http_code=None*, **args*, ***kwargs*)

Raised when the service is temporarily unavailable.

Parameters

- **`message`** (*`string|bytes`*) – Error message.
- **`http_code`** – HTTP code.

`exception` `chemspipy.errors.`**`ChemSpiPyTimeoutError`**

Raised when an asynchronous request times out.

CHAPTER 4

Additional Notes

4.1 License

4.1.1 Authors

ChemSpiPy is developed and maintained by Matt Swain and community contributors.

See a full list of contributors on the GitHub Contributors page.

4.1.2 ChemSpiPy License

4.2 Contributing

Contributions of any kind are greatly appreciated!

4.2.1 Feedback

The Issue Tracker is the best place to post any feature ideas, requests and bug reports.

4.2.2 Contributing

If you are able to contribute changes yourself, just fork the source code on GitHub, make changes and file a pull request. All contributions are welcome, no matter how big or small.

Quick guide to contributing

1. Fork the ChemSpiPy repository on GitHub, then clone your fork to your local machine:

```
git clone https://github.com/<username>/ChemSpiPy.git
cd ChemSpiPy
```

2. Install the development requirements into a conda environment:

```
conda env create -n chemspipy -f environment.yml
source activate chemspipy
```

3. Create a new branch for your changes:

```
git checkout -b <name-for-changes>
```

4. Make your changes or additions. Ideally add some tests and ensure they pass by running:

```
pytest
```

5. Commit your changes and push to your fork on GitHub:

```
git add .
git commit -m "<description-of-changes>"
git push origin <name-for-changes>
```

4. Submit a pull request.

Tips

- Follow the PEP8 style guide.
- Include docstrings as described in PEP257.
- Try and include tests that cover your changes.
- Try to write good commit messages.
- Read the GitHub help page on Using pull requests.

4.3 Migration Guide

4.3.1 Upgrading to version 2.x

The RSC released an entirely new REST API in 2018, necessitating a number of changes to ChemSpiPy. Where possible, backwards compatibility has been maintained, but many methods are deprecated and some have been removed entirely.

ChemSpider Object

- Instantiate the *`ChemSpider`* with a required `api_key` parameter instead of the optional `security_token` parameter.
- Deprecated methods:
  - `get_databases` → *`get_datasources()`*
  - `get_extended_compound_info` → *`get_details()`*
  - `get_extended_compound_info_list` → *`get_details_batch()`*
  - `get_extended_mol_compound_info_list` → *`get_details_batch()`*
  - `get_record_mol` → *`get_mol()`*
  - `async_simple_search` → *`filter_name()`*
  - `async_simple_search_ordered` → *`filter_name()`*
  - `get_async_search_status` → *`filter_status()`*
  - `get_async_search_status_and_count` → *`filter_status()`*
  - `get_async_search_result` → *`filter_results()`*
  - `get_async_search_result_part` → *`filter_results()`*
  - `get_compound_info` → *`get_details()`*
  - `get_compound_thumbnail` → *`get_image()`*
  - `simple_search` → *`search()`*
- Removed methods:
  - `get_original_mol`
  - `get_all_spectra_info`
  - `get_spectrum_info`
  - `get_compound_spectra_info`
  - `get_spectra_info_list`

Compound Object

- Non-standard InChI and InChIKey are no longer available. All are now 'standard'. Deprecated properties:
  - `stdinchi` → *`inchi`*
  - `stdinchikey` → *`inchikey`*
- Removed properties:

  - `xlogp`
  - `alogp`
  - `mol_3d`
  - `mol_raw`

Spectrum Object

- `Spectrum` object has been removed entirely.

`api` Module

- Removed `DIMENSIONS` mapping.
- Replaced *`FIELDS`* mapping with a list of available properties fields.
- Removed `xml_to_dict` function.

4.4 Change Log

4.4.1 v2.0.0 (2018-09-09)

Full Changelog

Implemented enhancements:

- Access "data source" information through `ChemSpider` #3
- Switch to new RSC REST API #12 (mcs07)

Fixed bugs:

- Authentication problem #11

Closed issues:

- Make conda-forge recipe #13
- return # of data sources in search results #10

Merged pull requests:

- Update docs for new RSC REST API #14 (mcs07)

4.4.2 v1.0.5 (2017-03-29)

Full Changelog

Implemented enhancements:

- Switch to pytest #7 (mcs07)

Fixed bugs:

- Add support for https #5
- Use https by default - fixes #5 #6 (mcs07)

Merged pull requests:

- Improve handling of invalid tokens #4 (mcs07)

4.4.3 v1.0.4 (2015-06-13)

Full Changelog

4.4.4 v1.0.3 (2015-03-05)

Full Changelog

4.4.5 v1.0.2 (2015-03-04)

Full Changelog

4.4.6 v1.0.1 (2014-09-15)

Implemented enhancements:

- Fix for UTF-8 encoding error (and some other tweaks)… #2 (nickfyson)

CHAPTER 5

Useful links

- ChemSpiPy on GitHub
- ChemSpiPy on PyPI
- Issue tracker
- Release history
- ChemSpiPy Travis CI

Python Module Index

Index