CIRpy Documentation

Release 1.0.2

Matt Swain

January 04, 2016

Contents

CIRpy is a Python interface for the Chemical Identifier Resolver (CIR) (http://cactus.nci.nih.gov/chemical/structure) by the CADD Group at the NCI/NIH.

CIR is a web service that will resolve any chemical identifier to another chemical representation. For example, you can pass it a chemical name and and request the corresponding SMILES string:

```
>>> import cirpy
>>> cirpy.resolve('Aspirin', 'smiles')
'C1=CC=CC(=C1C(O)=O)OC(C)=O'
```

CIRpy makes interacting with CIR through Python easy. There's no need to construct url requests and parse XML responses — CIRpy does all this for you.

CHAPTER 1

Features

- Resolve chemical identifiers such as names, CAS registry numbers, SMILES strings and SDF files to any other chemical representation.
- Get calculated properties such as molecular weight and hydrogen bond donor and acceptor counts.
- Download chemical file formats such as SDF, XYZ, CIF and CDXML.
- Get 2D compound depictions as a GIF or PNG images.
- Supports Python versions 2.7 – 3.4.
- Released under the MIT license (https://github.com/mcs07/CIRpy/blob/master/LICENSE).

CHAPTER 2

User guide

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

2.1 Installation

CIRpy supports Python versions 2.7, 3.3, 3.4 and 3.5. There are no required dependencies.

2.1.1 Option 1: Use pip (recommended)

The easiest and recommended way to install is using pip:

```
pip install cirpy
```

This will download the latest version of CIRpy, 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
```

2.1.2 Option 2: Download the latest release

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

```
tar -xzvf CIRpy-1.0.2.tar.gz
cd CIRpy-1.0.2
python setup.py install
```

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

2.1.3 Option 3: Clone the repository

The latest development version of CIRpy is always available on GitHub (https://github.com/mcs07/CIRpy). 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/CIRpy.git
cd CIRpy
python setup.py install
```

2.2 Getting started

This page gives a introduction on how to get started with CIRpy. Before we start, make sure you have *installed CIRpy* (page 5).

2.2.1 Basic usage

The simplest way to use CIRpy is with the `resolve` function:

```
>>> import cirpy
>>> cirpy.resolve('Aspirin', 'smiles')
'C1=CC=CC(=C1C(O)=O)OC(C)=O'
```

The first parameter is the input string and the second parameter is the desired output representation. The main output representations for the second parameter are:

```
stdinchi
stdinchikey
inchi
smiles
ficts
ficus
uuuuu
hashisy
sdf
names
iupac_name
cas
formula
```

All return a string, apart from `names` and `cas`, which return a list of strings.

2.2.2 File formats

Output can additionally be returned in a variety of file formats that are specified using the second parameter in the same way:

```
>>> cirpy.resolve('c1ccccc1', 'cif')
"data_C6H6\n#\n_chem_comp.id\t'C6H6'\n#\nloop_\n_chem_comp_atom.comp_id\n..."
```

The full list of file formats:

```
alc          # Alchemy format
cdxml        # CambridgeSoft ChemDraw XML format
cerius       # MSI Cerius II format
charmm       # Chemistry at HARvard Macromolecular Mechanics file format
cif          # Crystallographic Information File
cml          # Chemical Markup Language
ctx          # Gasteiger Clear Text format
gjf          # Gaussian input data file
gromacs      # GROMACS file format
hyperchem    # HyperChem file format
jme          # Java Molecule Editor format
maestro      # Schroedinger MacroModel structure file format
mol          # Symyx molecule file
mol2         # Tripos Sybyl MOL2 format
mrv          # ChemAxon MRV format
pdb          # Protein Data Bank
sdf3000      # Symyx Structure Data Format 3000
sln          # SYBYL Line Notation
xyz          # xyz file format
```

2.2.3 Properties

A number of calculated structure-based properties can be returned, also specified using the second parameter:

```
>>> cirpy.resolve('coumarin 343', 'h_bond_acceptor_count')
'5'
```

The full list of properties:

```
mw                          # (Molecular weight)
h_bond_donor_count
h_bond_acceptor_count
h_bond_center_count
rule_of_5_violation_count
rotor_count
effective_rotor_count
ring_count
ringsys_count
```

2.3 Resolvers

CIR interprets input strings using a series of "resolvers" in a specific order. Each one is tried in turn until one successfully interprets the input.

The available resolvers are not well documented, but the ones that I can identify, roughly in the order that they are tried by default, are:

```
smiles
stdinchikey
stdinchi
ncicadd_identifier      # (for FICTS, FICuS, uuuuu)
hashisy
cas_number
name_by_opsin
name_by_cir
```

2.3.1 Customizing resolvers

You can customize which resolvers are used (and the order they are used in), by supplying a list of resolvers as a third parameter to the `resolve` function:

```
>>> cirpy.resolve('Aspirin', 'sdf', ['cas_number', 'name_by_cir', 'name_by_opsin
'C9H8O4\nAPtclcactv03241513052D 0   0.00000     0.00000\n \n 21 21...'
>>> cirpy.resolve('C1=CC=CC(=C1C(O)=O)OC(C)=O', 'names', ['smiles', 'stdinchi'])
['2-acetyloxybenzoic acid', '2-Acetoxybenzoic acid', '50-78-2', ...]
```

Manually specifying the resolvers can be useful when an ambiguous input identifier could be interpreted as multiple different formats, but you know which format it is.

2.3.2 Resolving names

By default, CIR resolves names first by using OPSIN, and if that fails, using a lookup in its own name index. With CIRpy you can customize which of these resolvers are used, and also specify the order of precedence.

Just use the `resolve` function with a third parameter - a list containing any of the strings `name_by_opsin`, `name_by_cir` in the order in which they should be tried:

```
>>> cirpy.resolve('Morphine', 'smiles', ['name_by_opsin'])
'CN1CC[C@]23[C@H]4Oc5c(O)ccc(C[C@@H]1[C@@H]2C=C[C@@H]4O)c35'
>>> cirpy.resolve('Morphine', 'smiles', ['name_by_cir','name_by_opsin'])
'CN1CC[C@]23[C@H]4Oc5c(O)ccc(C[C@@H]1[C@@H]2C=CC4O)c35'
```

Read more about resolving names on the CIR blog (http://cactus.nci.nih.gov/blog/?p=1386).

Note: The `chemspider_id` and `name_by_chemspider` resolvers no longer exist.

2.4 Queries

The `resolve` function will only return the top match for a given input. However, sometimes multiple resolvers will match an input (e.g. the name resolvers), and individual resolvers can even return multiple results. The `query` function will return every result:

```
>>> cirpy.query('CCO', 'stdinchikey')
[Result(resolver='smiles', value='InChIKey=LFQSCWFLJHTTHZ-UHFFFAOYSA-N'), Result
```

As with the `resolve` function, it is possible to specify which resolvers are used:

```
>>> cirpy.query('2,4,6-trinitrotoluene', 'formula', ['name_by_opsin','name_by_ci
[Result(resolver='name_by_opsin', value='C7H5N3O6'), Result(resolver='name_by_ci
```

2.4.1 Results

The `query` function results a list of `Result` objects. Each `Result` has a `value` attribute that corresponds to what the `resolve` function would return:

```
>>> results = cirpy.query('2,4,6-trinitrotoluene', 'formula')
>>> results[0]
Result(resolver='name_by_opsin', value='C7H5N3O6')
>>> results[0].value
'C7H5N3O6'
```

Each `Result` also has `input`, `representation`, `resolver`, `input_format` and `notation` attributes. *See the full API documentation for information on these attributes* (page 13).

2.5 Miscellaneous

2.5.1 Tautomers

To get all possible resolved tautomers, use the `tautomers` parameter:

```
tautomers = query('warfarin', 'smiles', tautomers=True)
```

2.5.2 The Molecule object

The Molecule class provides an easy way to collect and store various structure representations and properties for a given input:

```
from cirpy import Molecule

mol = Molecule('N[C@@H](C)C(=O)O')
```

`mol` then has the following properties:

```
mol.stdinchi
mol.stdinchikey
mol.smiles
mol.ficts
mol.ficus
mol.uuuuu
mol.hashisy
mol.sdf
mol.names
mol.iupac_name
mol.cas
mol.image_url                # The url of a GIF image
mol.twirl_url                # The url of a TwirlyMol 3D viewer
mol.mw                       # Molecular weight
mol.formula
mol.h_bond_donor_count
mol.h_bond_acceptor_count
mol.h_bond_center_count
mol.rule_of_5_violation_count
mol.rotor_count
mol.effective_rotor_count
mol.ring_count
mol.ringsys_count
```

The first time you access each one of these properties, a request is made to the CIR servers. The result is cached, however, so subsequent access is much faster.

2.5.3 Downloading files

A convenience function is provided to facilitate downloading the CIR output to a file:

```
cirpy.download('Aspirin', 'test.sdf', 'sdf')
cirpy.download('Aspirin', 'test.sdf', 'sdf', overwrite=True)
```

This works in the same way as the `resolve` function, but also accepts a filename. There is an optional `overwrite` parameter to specify whether any existing file should be overwritten.

2.5.4 Constructing API URLs

Construct API URLs:

```
>>> cirpy.construct_api_url('Porphyrin', 'smiles')
'http://cactus.nci.nih.gov/chemical/structure/Porphyrin/smiles/xml'
```

2.5.5 Logging

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

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

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

2.5.6 Pattern matching

Note: It looks like the `name_pattern` resolver no longer works.

There is an additional `name_pattern` resolver that allows for Google-like searches. For example:

```
results = query('Morphine','smiles', ['name_pattern'])
```

The `notation` attribute of each `Result` will show you the name of the match (e.g. "Morphine N-oxide", "Morphine Sulfate") and the `value` attribute will be the representation specified in the query (SMILES in the above example).

Read more about pattern matching on the CIR blog (http://cactus.nci.nih.gov/blog/?p=1456).

2.6 Contributing

Contributions of any kind are greatly appreciated!

2.6.1 Feedback

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

2.6.2 Contributing

If you are able to contribute changes yourself, just fork the source code (https://github.com/mcs07/CIRpy) 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 CIRpy repository on GitHub (https://github.com/mcs07/CIRpy/fork), then clone your fork to your local machine:

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

2. Install the development requirements:

```
cd cirpy
pip install -r requirements/development.txt
```

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.
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 (https://github.com/mcs07/CIRpy/compare/).

Tips

- Follow the PEP8 (https://www.python.org/dev/peps/pep-0008) style guide.
- Include docstrings as described in PEP257 (https://www.python.org/dev/peps/pep-0257).
- Try and include tests that cover your changes.
- Try to write good commit messages (http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html).
- Consider squashing your commits (http://gitready.com/advanced/2009/02/10/squashing-commits-with-rebase.html) with rebase.
- Read the GitHub help page on Using pull requests (https://help.github.com/articles/using-pull-requests).

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 CIRpy source code and comments.

3.1.1 Resolve

cirpy.**resolve**(*input*, *representation*, *resolvers=None*, *get3d=False*, ***kwargs*)

Resolve input to the specified output representation.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Chemical identifier to resolve
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Desired output representation
- **resolvers** (*list(string)*) – (Optional) Ordered list of resolvers to use
- **get3d** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return 3D coordinates (where applicable)

Returns Output representation or None

Return type string or None

Raises

- **HTTPError** – if CIR returns an error code
- **ParseError** – if CIR response is uninterpretable

3.1.2 Query

cirpy.**query**(*input*, *representation*, *resolvers=None*, *get3d=False*, *tautomers=False*, ***kwargs*)

Get all results for resolving input to the specified output representation.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Chemical identifier to resolve
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Desired output representation
- **resolvers** (*list(string)*) – (Optional) Ordered list of resolvers to use
- **get3d** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return 3D coordinates (where applicable)
- **tautomers** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return all tautomers

Returns List of resolved results

Return type list(Result)

Raises

- **HTTPError** – if CIR returns an error code
- **ParseError** – if CIR response is uninterpretable

3.1.3 Result

class cirpy.**Result**(*input*, *notation*, *input_format*, *resolver*, *representation*, *value*)

A single result returned by CIR.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Originally supplied input identifier that produced this result
- **notation** (*string* (http://docs.python.org/library/string.html#module-string)) – Identifier matched by the resolver or tautomer ID
- **input_format** (*string* (http://docs.python.org/library/string.html#module-string)) – Format of the input as interpreted by the resolver
- **resolver** (*string* (http://docs.python.org/library/string.html#module-string)) – Resolver used to produce this result
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Requested output representation
- **value** (*string or list(string)*) – Actual result value

to_dict()
Return a dictionary containing Result data.

3.1.4 Images

cirpy.**resolve_image**(*input*, *resolvers=None*, *fmt=u'png'*, *width=300*, *height=300*, *frame=False*, *crop=None*, *bgcolor=None*, *atomcolor=None*, *hcolor=None*, *bondcolor=None*, *framecolor=None*, *symbolfontsize=11*, *linewidth=2*, *hsymbol=u'special'*, *csymbol=u'special'*, *stereolabels=False*, *stereowedges=True*, *header=None*, *footer=None*, ***kwargs*)
Resolve input to a 2D image depiction.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Chemical identifier to resolve
- **resolvers** (*list(string)*) – (Optional) Ordered list of resolvers to use
- **fmt** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) gif or png image format (default png)
- **width** (*int* (http://docs.python.org/library/functions.html#int)) – (Optional) Image width in pixels (default 300)
- **height** (*int* (http://docs.python.org/library/functions.html#int)) – (Optional) Image height in pixels (default 300)
- **frame** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to show border frame (default False)
- **crop** (*int* (http://docs.python.org/library/functions.html#int)) – (Optional) Crop image with specified padding
- **symbolfontsize** (*int* (http://docs.python.org/library/functions.html#int)) – (Optional) Atom label font size (default 11)
- **linewidth** (*int* (http://docs.python.org/library/functions.html#int)) – (Optional) Bond line width (default 2)
- **bgcolor** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Background color
- **atomcolor** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Atom label color
- **hcolor** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Hydrogen atom label color
- **bondcolor** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Bond color

- **framecolor** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Border frame color
- **hsymbol** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Hydrogens: all, special or none (default special)
- **csymbol** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Carbons: all, special or none (default special)
- **stereolabels** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to show stereochemistry labels (default False)
- **stereowedges** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to show wedge/dash bonds (default True)
- **header** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Header text above structure
- **footer** (*string* (http://docs.python.org/library/string.html#module-string)) – (Optional) Footer text below structure

3.1.5 Request

cirpy.**request**(*input*, *representation*, *resolvers=None*, *get3d=False*, *tautomers=False*, ***kwargs*)

Make a request to CIR and return the XML response.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Chemical identifier to resolve
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Desired output representation
- **resolvers** (*list(string)*) – (Optional) Ordered list of resolvers to use
- **get3d** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return 3D coordinates (where applicable)
- **tautomers** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return all tautomers

Returns XML response from CIR

Return type Element

Raises

- **HTTPError** – if CIR returns an error code
- **ParseError** – if CIR response is uninterpretable

3.1.6 Download

cirpy.**download**(*input*, *filename*, *representation*, *overwrite=False*, *resolvers=None*, *get3d=False*, ***kwargs*)

Convenience function to save a CIR response as a file.

This is just a simple wrapper around the resolve function.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Chemical identifier to resolve
- **filename** (*string* (http://docs.python.org/library/string.html#module-string)) – File path to save to
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Desired output representation
- **overwrite** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to allow overwriting of an existing file
- **resolvers** (*list(string)*) – (Optional) Ordered list of resolvers to use
- **get3d** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return 3D coordinates (where applicable)

Raises

- **HTTPError** – if CIR returns an error code
- **ParseError** – if CIR response is uninterpretable
- **IOError** – if overwrite is False and file already exists

3.1.7 API URLs

cirpy.**construct_api_url**(*input*, *representation*, *resolvers=None*, *get3d=False*, *tautomers=False*, *xml=True*, ***kwargs*)

Return the URL for the desired API endpoint.

Parameters

- **input** (*string* (http://docs.python.org/library/string.html#module-string)) – Chemical identifier to resolve
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Desired output representation
- **resolvers** (*list(str)*) – (Optional) Ordered list of resolvers to use
- **get3d** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return 3D coordinates (where applicable)

- **tautomers** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return all tautomers
- **xml** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to return full XML response

Returns CIR API URL

Return type str (http://docs.python.org/library/functions.html#str)

3.1.8 Molecule

class cirpy.**Molecule**(*input*, *resolvers=None*, *get3d=False*, ***kwargs*)

Class to hold and cache the structure information for a given CIR input.

Initialize with a resolver input.

stdinchi
Standard InChI.

stdinchikey
Standard InChIKey.

inchi
Non-standard InChI. (Uses options DONOTADDH W0 FIXEDH RECMET NEWPS SPXYZ SAsXYZ Fb Fnud).

smiles
SMILES string.

ficts
FICTS NCI/CADD hashed structure identifier.

ficus
FICuS NCI/CADD hashed structure identifier.

uuuuu
uuuuu NCI/CADD hashed structure identifier.

hashisy
CACTVS HASHISY identifier.

sdf
SDF file.

names
List of chemical names.

iupac_name
IUPAC approved name.

cas
CAS registry numbers.

mw
Molecular weight.

formula
Molecular formula

h_bond_donor_count
Hydrogen bond donor count.

h_bond_acceptor_count
Hydrogen bond acceptor count.

h_bond_center_count
Hydrogen bond center count.

rule_of_5_violation_count
Rule of 5 violation count.

rotor_count
Rotor count.

effective_rotor_count
Effective rotor count.

ring_count
Ring count.

ringsys_count
Ring system count.

image
2D image depiction.

image_url
URL of a GIF image.

twirl_url
Url of a TwirlyMol 3D viewer.

download(*filename*, *representation*, *overwrite=False*)
Download the resolved structure as a file.

Parameters

- **filename** (*string* (http://docs.python.org/library/string.html#module-string)) – File path to save to
- **representation** (*string* (http://docs.python.org/library/string.html#module-string)) – Desired output representation
- **overwrite** (*bool* (http://docs.python.org/library/functions.html#bool)) – (Optional) Whether to allow overwriting of an existing file