pypath Documentation

Release 0.10.6

Dénes Türei

Feb 25, 2020

CONTENTS:

note `pypath` supports both Python 2.7 and Python 3.6+. In the beginning, pypath has been developed only for Python 2.7. Then the code have been adjusted to Py3 and for a few years we develop and test `pypath` in Python 3. Therefore this is the better supported Python variant.

documentation http://saezlab.github.io/pypath

issues https://github.com/saezlab/pypath/issues

CHAPTER

ONE

INSTALLATION

1.1 Linux

In almost any up-to-date Linux distribution the dependencies of **pypath** are built-in, or provided by the distributors. You only need to install a couple of things in your package manager (cairo, py(2)cairo, igraph, python(2)-igraph, graphviz, pygraphviz), and after install **pypath** by *pip* (see below). If any module still missing, you can install them the usual way by *pip* or your package manager.

1.1.1 igraph C library, cairo and pycairo

python(2)-igraph is a Python interface to use the igraph C library. The C library must be installed. The same goes for *cairo*, *py(2)cairo* and *graphviz*.

1.1.2 Directly from git

```
pip install git+https://github.com/saezlab/pypath.git
```

1.1.3 With pip

Download the package from /dist, and install with pip:

```
pip install pypath-x.y.z.tar.gz
```

1.1.4 Build source distribution

Clone the git repo, and run setup.py:

```
python setup.py sdist
```

1.2 Mac OS X

On OS X installation is not straightforward primarily because cairo needs to be compiled from source. We provide 2 scripts here: the **mac-install-brew.sh** installs everything with HomeBrew, and **mac-install-conda.sh** installs from Anaconda distribution. With these scripts installation of igraph, cairo and graphviz goes smoothly most of the time, and options are available for omitting the 2 latter. To know more see the description in the script header. There is a

third script **mac-install-source.sh** which compiles everything from source and presumes only Python 2.7 and Xcode installed. We do not recommend this as it is time consuming and troubleshooting requires expertise.

1.2.1 Troubleshooting

- `no module named ...` when you try to load a module in Python. Did theinstallation of the module run without error? Try to run again the specific part from the mac install shell script to see if any error comes up. Is the path where the module has been installed in your `$PYTHONPATH`? Try `echo $PYTHONPATH` to see the current paths. Add your local install directories if those are not there, e.g. `export PYTHONPATH="/Users/me/local/python2.7/site-packages:$PYTHONPATH"`. If it works afterwards, don't forget to append these export path statements to your `~/.bash_profile`, so these will be set every time you launch a new shell.
- `pkgconfig` not found. Check if the `$PKG_CONFIG_PATH` variable is set correctly, and pointing on a directory where pkgconfig really can be found.
- Error while trying to install py(2)cairo by pip. py(2)cairo could not be installed by pip, but only by waf. Please set the `$PKG_CONFIG_PATH` before. See **mac-install-source.sh** on how to install with waf.
- Error at pygraphviz build: `graphviz/cgraph.h file not found`. This is because the directory of graphviz detected wrong by pkgconfig. See **mac-install-source.sh** how to set include dirs and library dirs by `--global-option` parameters.
- Can not install bioservices, because installation of jurko-suds fails. Ok, this fails because pip is not able to install the recent version of setuptools, because a very old version present in the system path. The development version of jurko-suds does not require setuptools, so you can install it directly from git as it is done in **mac-install-source.sh**.
- In **Anaconda**, *pypath* can be imported, but the modules and classes are missing. Apparently Anaconda has some built-in stuff called *pypath*. This has nothing to do with this module. Please be aware that Anaconda installs a completely separated Python distribution, and does not detect modules in the main Python installation. You need to install all modules within Anaconda's directory. **mac-install-conda.sh** does exactly this. If you still experience issues, please contact us.

1.3 Microsoft Windows

Not many people have used *pypath* on Microsoft computers so far. Please share your experiences and contact us if you encounter any issue. We appreciate your feedback, and it would be nice to have better support for other computer systems.

1.3.1 With Anaconda

The same workflow like you see in `mac-install-conda.sh` should work for Anaconda on Windows. The only problem you certainly will encounter is that not all the channels have packages for all platforms. If certain channel provides no package for Windows, or for your Python version, you just need to find an other one. For this, do a search:

```
anaconda search -t conda <package name>
```

For example, if you search for *pycairo*, you will find out that *vgauther* provides it for osx-64, but only for Python 3.4, while *richlewis* provides also for Python 3.5. And for win-64 platform, there is the channel of *KristanAmstrong*. Go along all the commands in `mac-install-conda.sh`, and modify the channel if necessary, until all packages install successfully.

1.3.2 With other Python distributions

Here the basic principles are the same as everywhere: first try to install all external dependencies, after *pip* install should work. On Windows certain packages can not be installed by compiled from source by *pip*, instead the easiest to install them precompiled. These are in our case *fisher, lxml, numpy (mkl version), pycairo, igraph, pygraphviz, scipy and statsmodels*. The precompiled packages are available here: http://www.lfd.uci.edu/~gohlke/pythonlibs/. We tested the setup with Python 3.4.3 and Python 2.7.11. The former should just work fine, while with the latter we have issues to be resolved.

1.3.3 Known issues

- *"No module fabric available."* – or *pysftp* missing: this is not

important, only certain data download methods rely on these modules, but likely you won't call those at all. * Progress indicator floods terminal: sorry about that, will be fixed soon. * Encoding related exceptions in Python2: these might occur at some points in the module, please send the traceback if you encounter one, and we will fix as soon as possible.

Special thanks to Jorge Ferreira for testing pypath on Windows!

CHAPTER

TWO

REFERENCE

2.1 annot_formats

2.2 annot

class pypath.core.annot.**Adhesome**(***kwargs*)

class pypath.core.annot.**AnnotationBase**(*name*, *ncbi_tax_id=9606*, *input_method=None*, *input_args=None*, *entity_type='protein'*, *swissprot_only=True*, *proteins=()*, *complexes=()*, *reference_set=()*, *infer_complexes=True*, *dump=None*, ***kwargs*)

add_complexes_by_inference(*complexes=None*)
Creates complex annotations by in silico inference and adds them to this annotation set.

all_proteins()
All UniProt IDs annotated in this resource.

annotate_complex(*cplex*)
Infers annotations for a single complex.

complex_inference(*complexes=None*)
Annotates all complexes in *complexes*, by default in the default complex database (existing in the *complex* module or generated on demand according to the module's current settings).

Dict with complexes as keys and sets of annotations as values. Complexes with no valid information in this annotation resource won't be in the dict.

complexes [iterable] Iterable yielding complexes.

get_subset(*method=None*, ***kwargs*)
Retrieves a subset by filtering based on `kwargs`. Each argument should be a name and a value or set of values. Elements having the provided values in the annotation will be returned. Returns a set of UniProt IDs.

load_proteins()
Retrieves a set of all UniProt IDs to have a base set of the entire proteome.

reload()
Reloads the object from the module level.

class pypath.core.annot.**Baccin2019**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**CancerGeneCensus**(***kwargs*)

class pypath.core.annot.**Cancersea**(***kwargs*)

class pypath.core.annot.**CellPhoneDB**(***kwargs*)

record
alias of *pypath.inputs.main.CellPhoneDBAnnotation*

class pypath.core.annot.**CellPhoneDBComplex**(***kwargs*)

class pypath.core.annot.**CellSurfaceProteinAtlas**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Comppi**(***kwargs*)

class pypath.core.annot.**Corum**(*name*, *annot_attr*, ***kwargs*)

class pypath.core.annot.**CorumFuncat**(***kwargs*)

class pypath.core.annot.**CorumGO**(***kwargs*)

class pypath.core.annot.**Cpad**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Dgidb**(***kwargs*)

class pypath.core.annot.**Disgenet**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Exocarta**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**GOIntercell**(*categories=None*, *go_annot=None*, *ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**GuideToPharmacology**(*load_sources=False*, ***kwargs*)

class pypath.core.annot.**Hgnc**(***kwargs*)

class pypath.core.annot.**HpmrComplex**(***kwargs*)

class pypath.core.annot.**HumanPlasmaMembraneReceptome**(***kwargs*)

class pypath.core.annot.**HumanProteinAtlas**(***kwargs*)

class pypath.core.annot.**HumanProteinAtlasSecretome**(***kwargs*)

class pypath.core.annot.**HumanProteinAtlasSubcellular**(***kwargs*)

class pypath.core.annot.**Integrins**(***kwargs*)

class pypath.core.annot.**Intogen**(***kwargs*)

class pypath.core.annot.**KeggPathways**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Kinasedotcom**(***kwargs*)

class pypath.core.annot.**Kirouac2010**(*load_sources=False*, ***kwargs*)

class pypath.core.annot.**LigandReceptor**(*name*, *ligand_col=None*, *receptor_col=None*, *ligand_id_type=None*, *receptor_id_type=None*, *record_processor_method=None*, *record_extra_fields=None*, *record_defaults=None*, *extra_fields_methods=None*, ***kwargs*)

class pypath.core.annot.**Locate**(*ncbi_tax_id=9606*, *literature=True*, *external=True*, *predictions=False*, ***kwargs*)

class pypath.core.annot.**Lrdb**(***kwargs*)

class pypath.core.annot.**Matrisome**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Matrixdb**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Membranome**(***kwargs*)

class pypath.core.annot.**Msigdb**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**NetpathPathways**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Opm**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Phosphatome**(***kwargs*)

class pypath.core.annot.**Ramilowski2015**(*load_sources=False*, ***kwargs*)

class pypath.core.annot.**Ramilowski2015Location**(***kwargs*)

class pypath.core.annot.**SignalinkPathways**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**SignorPathways**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Surfaceome**(***kwargs*)

class pypath.core.annot.**Tfcensus**(***kwargs*)

class pypath.core.annot.**Topdb**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Vesiclepedia**(*ncbi_tax_id=9606*, ***kwargs*)

class pypath.core.annot.**Zhong2015**(***kwargs*)

pypath.core.annot.**get_db**(*keep_annotators=True*, *create_dataframe=False*, *use_complexes=True*, ***kwargs*)

Retrieves the current database instance and initializes it if does not exist yet.

pypath.core.annot.**init_db**(*keep_annotators=True*, *create_dataframe=False*, *use_complexes=True*, ***kwargs*)

Initializes or reloads the annotation database. The database will be assigned to the `db` attribute of this module.

2.3 bel

2.4 cache

pypath.share.cache.**get_cachedir**(*cachedir=None*)

Ensures the cache directory exists and returns its path.

2.5 cellphonedb

2.6 common

pypath.share.common.**uniq_list**(*seq*)

Reduces a list to its unique elements.

Takes any iterable and returns a list of unique elements on it. If the argument is a dictionary, returns a list of unique keys. **NOTE:** Does not preserve the order of the elements.

Parameters **seq** (*list*) – Sequence to be processed, can be any iterable type.

Returns (*list*) – List of unique elements in the sequence *seq*.

Examples:

```
>>> uniq_list('aba')
['a', 'b']
>>> uniq_list([0, 1, 2, 1, 0])
[0, 1, 2]
```

pypath.share.common.**add_to_list**(*lst*, *toadd*)

Adds elements to a list.

Appends *toadd* to *lst*. Function differs from `list.append()` since is capable to handle different data types. This is, if *lst* is not a list, it will be converted to one. Similarly, if *toadd* is not a list, it will be converted and added. If *toadd* is or contains `None`, these will be ommited. The returned list will only contain unique elements and does not necessarily preserve order.

Parameters

- **lst** (*list*) – List or any other type (will be converted into a list). Original sequence to which *toadd* will be appended.
- **toadd** (*any*) – Element(s) to be added into *lst*.

Returns (*list*) – Contains the unique element(s) from the union of *lst* and *toadd*. **NOTE:** Makes use of `common.uniq_list()`, does not preserve order of elements.

Examples:

```
>>> add_to_list('ab', 'cd')
['ab', 'cd']
>>> add_to_list('ab', ['cd', None, 'ab', 'ef'])
['ab', 'ef', 'cd']
>>> add_to_list((0, 1, 2), 4)
[0, 1, 2, 4]
```

pypath.share.common.**add_to_set**(*st*, *toadd*)

Adds elements to a set.

Appends *toadd* to *st*. Function is capable to handle different input data types. This is, if *toadd* is a list, it will be converted to a set and added.

Parameters

- **st** (*set*) – Original set to which *toadd* will be added.
- **toadd** (*any*) – Element(s) to be added into *st*.

Returns (*set*) – Contains the element(s) from the union of *st* and *toadd*.

Examples:

```
>>> st = set([0, 1, 2])
>>> add_to_set(st, 3)
set([0, 1, 2, 3])
>>> add_to_set(st, [4, 2, 5])
set([0, 1, 2, 4, 5])
```

pypath.share.common.**gen_session_id**(*length=5*)

Generates a random alphanumeric string.

Parameters **length** (*int*) – Optional, `5` by default. Specifies the length of the random string.

Returns (*str*) – Random alphanumeric string of the specified length.

pypath.share.common.**sorensen_index**(*a*, *b*)
Computes the Sorensen index.

Computes the Sorensen-Dice coefficient between two sets *a* and *b*.

Parameters

- **a** (*set*) – Or any iterable type (will be converted to set).
- **b** (*set*) – Or any iterable type (will be converted to set).

Returns (*float*) – The Sorensen-Dice coefficient between *a* and *b*.

pypath.share.common.**simpson_index**(*a*, *b*)
Computes Simpson's index.

Given two sets *a* and *b*, returns the Simpson index.

Parameters

- **a** (*set*) – Or any iterable type (will be converted to set).
- **b** (*set*) – Or any iterable type (will be converted to set).

Returns (*float*) – The Simpson index between *a* and *b*.

pypath.share.common.**simpson_index_counts**(*a*, *b*, *c*)

Parameters

- **a** –
- **b** –
- **c** –

Returns (*float*) –

pypath.share.common.**jaccard_index**(*a*, *b*)
Computes the Jaccard index.

Computes the Jaccard index between two sets *a* and *b*.

Parameters

- **a** (*set*) – Or any iterable type (will be converted to set).
- **b** (*set*) – Or any iterable type (will be converted to set).

Returns (*float*) – The Jaccard index between *a* and *b*.

pypath.share.common.**console**(*message*)
Prints a message on the terminal.

Prints a *message* to the standard output (e.g. terminal) formatted to 80 characters per line plus first-level indentation.

Parameters **message** (*str*) – The message to be printed.

pypath.share.common.**wcl**(*f*)

pypath.share.common.**flat_list**(*lst*)
Coerces the elements of a list of iterables into a single list.

Parameters **lst** (*lsit*) – List to be flattened. Its elements can be also lists or any other iterable.

Returns (*list*) – Flattened list of *lst*.

Examples:

```
>>> flat_list([(0, 1), (1, 1), (2, 1)])
[0, 1, 1, 1, 2, 1]
>> flat_list(['abc', 'def'])
['a', 'b', 'c', 'd', 'e', 'f']
```

pypath.share.common.**del_empty**(*lst*)

Removes empty entries of a list.

It is assumed that elemenst of *lst* are iterables (e.g. [str] or [list]).

Parameters **lst** (*list*) – List from which empty elements will be removed.

Returns (*list*) – Copy of *lst* without elements whose length was zero.

Example:

```
>>> del_empty(['a', '', 'b', 'c'])
['a', 'b', 'c']
```

pypath.share.common.**get_args**(*loc_dict*, *remove={}*)

Given a dictionary of local variables, returns a copy of it without `'self'`, `'kwargs'` (in the scope of a `class`) plus any other specified in the keyword argument *remove*.

Parameters

- **loc_dict** (*dict*) – Dictionary containing the local variables (e.g. a call to `locals()` in a given scope).
- **remove** (*set*) – Optional, `set([])` by default. Can also be a list. Contains the keys of the elements in *loc_dict* that will be removed.

Returns (*dict*) – Copy of *loc_dict* without `'self'`, `'kwargs'` and any other element specified in *remove*.

pypath.share.common.**something**(*anything*)

Checks if argument is empty.

Checks if *anything* is empty or `None`.

Parameters **anything** (*any*) – Self-explanatory.

Returns (*bool*) – `False` if *anyhting* is `None` or any empty data type.

Examples:

```
>>> something(None)
False
>>> something(123)
True
>>> something('Hello world!')
True
>>> something('')
False
>>> something([])
False
```

pypath.share.common.**rotate**(*point*, *angle*, *center=(0.0, 0.0)*)
Rotates a point with respect to a center.

Rotates a given *point* around a *center* according to the specified *angle* (in degrees) in a two-dimensional space. The rotation is counter-clockwise.

Parameters

- **point** (*tuple*) – Or list. Contains the two coordinates of the point to be rotated.
- **angle** (*float*) – Angle (in degrees) from which the point will be rotated with respect to *center* (counter-clockwise).
- **center** (*tuple*) – Optional, `(0.0, 0.0)` by default. Can also be a list. Determines the two coordinates of the center relative to which the point has to be rotated.

Returns (*tuple*) – Pair of coordinates of the rotated point.

pypath.share.common.**clean_dict**(*dct*)
Cleans a dictionary of `None` values.

Removes `None` values from a dictionary *dct* and casts all other values to strings.

Parameters **dct** (*dict*) – Dictionary to be cleaned from `None` values.

Returns (*dict*) – Copy of *dct* without `None` value entries and all other values formatted to strings.

pypath.share.common.**md5**(*value*)
Computes the sum of MD5 hash of a given string *value*.

Parameters **value** (*str*) – Or any other type (will be converted to string). Value for which the MD5 sum will be computed. Must follow ASCII encoding.

Returns (*str*) – Hash value resulting from the MD5 sum of the *value* string.

pypath.share.common.**uniq_ord_list**(*seq*, *idfun=None*)
Reduces a list to its unique elements keeping their order.

Returns a copy of *seq* without repeated elements. Preserves the order. If any element is repeated, the first instance is kept.

Parameters

- **seq** (*list*) – Or any other iterable type. The sequence from which repeated elements are to be removed.
- **idfun** (*function*) – Optional, `None` by default. Identifier function, for each entry of *seq*, returns a identifier of that entry from which uniqueness is determined. Default behavior is f(x) = x. See examples below.

Returns (*list*) – Copy of *seq* without the repeated elements (according to *idfun*).

Examples:

```
>>> uniq_ord_list([0, 1, 2, 1, 5])
[0, 1, 2, 5]
>>> uniq_ord_list('abracadabra')
['a', 'b', 'r', 'c', 'd']
>>> def f(x):
...     if x > 0:
...             return 0
...     else:
...             return 1
>>> uniq_ord_list([-32, -42, 1, 15, -12], idfun=f)
```

(continues on next page)

(continued from previous page)

```
[-32, 1]
>>> def g(x): # Given a file name, return it without extension
...     return x.split('.')[0]
>>> uniq_ord_list(['a.png', 'a.txt', 'b.pdf'], idfun=g)
['a.png', 'b.pdf']
```

pypath.share.common.**dict_diff**(*d1*, *d2*)

Compares two dictionaries.

Compares two given dictionaries *d1* and *d2* whose values are sets or dictionaries (in such case the function is called recursively). **NOTE:** The comparison is only performed on the values of the keys that are common in *d1* and *d2* (see example below).

Parameters

- **d1** (*dict*) – First dictionary of the comparison.
- **d2** (*dict*) – Second dictionary of the comparison.

Returns

- (*dict*) – Unique elements of *d1* when compared to *d2*.
- (*dict*) – Unique elements of *d2* when compared to *d1*.

Examples:

```
>>> d1 = {'a': {1}, 'b': {2}, 'c': {3}} # 'c' is unique to d1
>>> d2 = {'a': {1}, 'b': {3}}
>>> dict_diff(d1, d2)
({'a': set([]), 'b': set([2])}, {'a': set([2]), 'b': set([3])})
```

pypath.share.common.**to_set**(*var*)

Makes sure the object *var* is a set, if it is a list converts it to set, otherwise it creates a single element set out of it. If *var* is None returns empty set.

pypath.share.common.**to_list**(*var*)

Makes sure *var* is a list otherwise creates a single element list out of it. If *var* is None returns empty list.

pypath.share.common.**unique_list**(*seq*)

Reduces a list to its unique elements.

Takes any iterable and returns a list of unique elements on it. If the argument is a dictionary, returns a list of unique keys. **NOTE:** Does not preserve the order of the elements.

Parameters **seq** (*list*) – Sequence to be processed, can be any iterable type.

Returns (*list*) – List of unique elements in the sequence *seq*.

Examples:

```
>>> uniq_list('aba')
['a', 'b']
>>> uniq_list([0, 1, 2, 1, 0])
[0, 1, 2]
```

pypath.share.common.**basestring**

alias of builtins.str

pypath.share.common.**is_float**(*num*)
Tells if a string represents a floating point number, i.e. it can be converted by *float*.

pypath.share.common.**is_int**(*num*)
Tells if a string represents an integer, i.e. it can be converted by *int*.

pypath.share.common.**float_or_nan**(*num*)
Returns *num* converted from string to float if *num* represents a float otherwise *numpy.nan*.

2.7 complex

class pypath.core.complex.**AbstractComplexResource**(*name*, *ncbi_tax_id=9606*, *input_method=None*, *input_args=None*, *dump=None*, ***kwargs*)
A resource which provides information about molecular complexes.

class pypath.core.complex.**CellPhoneDB**(***kwargs*)

class pypath.core.complex.**Compleat**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**ComplexAggregator**(*resources=None*, *pickle_file=None*)

reload()
Reloads the object from the module level.

class pypath.core.complex.**ComplexPortal**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**Corum**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**GuideToPharmacology**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**Havugimana**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**Hpmr**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**Humap**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**Pdb**(*input_args=None*, ***kwargs*)

class pypath.core.complex.**Signor**(*input_args=None*, ***kwargs*)

pypath.core.complex.**all_complexes**()
Returns a set of all complexes in the database which serves as a reference set for many methods, just like `inputs.uniprot.all_uniprots` represents the proteome.

pypath.core.complex.**get_db**(***kwargs*)
Retrieves the current database instance and initializes it if does not exist yet.

pypath.core.complex.**init_db**(***kwargs*)
Initializes or reloads the complex database. The database will be assigned to the `db` attribute of this module.

2.8 curl

class pypath.share.curl.**Curl**(*url*, *silent=True*, *get=None*, *post=None*, *req_headers=None*, *cache=True*, *debug=False*, *outf=None*, *compr=None*, *encoding=None*, *files_needed=None*, *connect_timeout=300*, *timeout=2400*, *ignore_content_length=False*, *init_url=None*, *init_fun='get_jsessionid'*, *init_use_cache=False*, *follow=True*, *large=False*, *default_mode='r'*, *override_post=False*, *init_headers=False*, *return_headers=False*, *compressed=False*, *binary_data=None*, *write_cache=True*, *force_quote=False*, *sftp_user=None*, *sftp_passwd=None*, *sftp_passwd_file='.secrets'*, *sftp_port=22*, *sftp_host=None*, *sftp_ask=None*, *setup=True*, *call=True*, *process=True*, *retries=3*, *cache_dir=None*, *bypass_url_encoding=False*, *empty_attempt_again=True*)

This class is a wrapper around pycurl. You can set a vast amount of parameters. In addition it has a caching functionality: using this downloads of databases/resources is performed only once. It handles HTTP, FTP, cookies, headers, GET and POST params, multipart/form data, URL quoting, redirects, timeouts, retries, encodings, debugging. It returns either downloaded data, file pointer, files extracted from archives (gzip, tar.gz, zip). It is able to show a progress and status indicator on the console.

close()
Closes all file objects.

construct_binary_data()
The binary data content of a *form/multipart* type request can be constructed from a list of tuples (<field name>, <field value>), where field name and value are both type of bytes.

is_quoted(*string*)
From http://stackoverflow.com/questions/ 1637762/test-if-string-is-url-encoded-in-php

set_binary_data()
Set binary data to be transmitted attached to POST request.

binary_data is either a bytes string, or a filename, or a list of key-value pairs of a multipart form.

url_fix(*charset='utf-8'*)
From http://stackoverflow.com/a/121017/854988

class pypath.share.curl.**FileOpener**(*file_param*, *compr=None*, *extract=True*, *_open=True*, *set_fileobj=True*, *files_needed=None*, *large=True*, *default_mode='r'*, *encoding='utf-8'*)

This class opens a file, extracts it in case it is a gzip, tar.gz, tar.bz2 or zip archive, selects the requested files if you only need certain files from a multifile archive, reads the data from the file, or returns the file pointer, as you request. It examines the file type and size.

extract()
Calls the extracting method for compressed files.

open()
Opens the file if exists.

open_tgz()
Extracts files from tar gz.

class pypath.share.curl.**cache_delete_off**
This is a context handler which stops pypath.curl.Curl() deleting the cache files. This is the default behaviour, so this context won't change anything by default.

Behind the scenes it sets the value of the *pypath.curl.CACHEDEL* module level variable to *False*.

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_delete_off():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**cache_delete_on**

This is a context handler which results pypath.curl.Curl() deleting the cache files instead of reading it. Then it downloads the data again, or does nothing if the *DRYRUN* context is turned on. Upon deleting cache files console messages will let you know which files have been deleted.

Behind the scenes it sets the value of the *pypath.curl.CACHEDEL* module level variable to *True* (by default it is *False*).

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_delete_on():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**cache_off**

This is a context handler to turn off pypath.curl.Curl() cache. Data will be downloaded even if it exists in cache.

Behind the scenes it sets the value of the *pypath.curl.CACHE* module level variable to *False* (by default it is *None*).

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

print('`curl.CACHE` is ', curl.CACHE)

with curl.cache_on():
    print('`curl.CACHE` is ', curl.CACHE)
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**cache_on**

This is a context handler to turn on pypath.curl.Curl() cache. As most of the methods use cache as their default behaviour, probably it won't change anything.

Behind the scenes it sets the value of the *pypath.curl.CACHE* module level variable to *True* (by default it is *None*).

Example:

```
import pypath
from pypath import curl, data_formats

```

(continues on next page)

(continued from previous page)

```
pa = pypath.PyPath()

print('`curl.CACHE` is ', curl.CACHE)

with curl.cache_on():
    print('`curl.CACHE` is ', curl.CACHE)
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**cache_print_off**

This is a context handler which stops pypath.curl.Curl() to print verbose messages about its cache.

Behind the scenes it sets the value of the *pypath.curl.CACHEPRINT* module level variable to *False*. As by default it is *False*, this context won't modify the default behaviour.

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_print_off():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**cache_print_on**

This is a context handler which makes pypath.curl.Curl() print verbose messages about its cache.

Behind the scenes it sets the value of the *pypath.curl.CACHEPRINT* module level variable to *True* (by default it is *False*).

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_print_on():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**debug_off**

This is a context handler which avoids pypath.curl.Curl() to print debug information. By default it does not do this, so this context only restores the default.

Behind the scenes it sets the value of the *pypath.curl.DEBUG* module level variable to *False*.

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_debug_off():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**debug_on**

This is a context handler which results pypath.curl.Curl() to print debug information. This is useful if you have some issue with *Curl*, and you want to see what's going on.

Behind the scenes it sets the value of the *pypath.curl.DEBUG* module level variable to *True* (by default it is *False*).

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_debug_on():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**dryrun_off**

This is a context handler which results pypath.curl.Curl() to perform download or cache read. This is the default behaviour, so applying this context restores the default.

Behind the scenes it sets the value of the *pypath.curl.DRYRUN* module level variable to *False*.

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_dryrun_off():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**dryrun_on**

This is a context handler which results pypath.curl.Curl() to do all setup steps, but do not perform download or cache read.

Behind the scenes it sets the value of the *pypath.curl.DRYRUN* module level variable to *True* (by default it is *False*).

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_dryrun_on():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**preserve_off**

This is a context handler which avoids pypath.curl.Curl() to make a reference to itself in the module level variable *LASTCURL*. By default it does not do this, so this context only restores the default.

Behind the scenes it sets the value of the *pypath.curl.PRESERVE* module level variable to *False*.

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()
```

(continues on next page)

(continued from previous page)

```
with curl.cache_preserve_off():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

class pypath.share.curl.**preserve_on**

This is a context handler which results pypath.curl.Curl() to make a reference to itself in the module level variable *LASTCURL*. This is useful if you have some issue with *Curl*, and you want to access the instance for debugging.

Behind the scenes it sets the value of the *pypath.curl.PRESERVE* module level variable to *True* (by default it is *False*).

Example:

```
import pypath
from pypath import curl, data_formats

pa = pypath.PyPath()

with curl.cache_preserve_on():
    pa.load_resources({'signor': data_formats.pathway['signor']})
```

2.9 data_formats

pypath.resources.data_formats.**interaction = {'alz': <pypath.internals.input_formats.Netwo**

PTM databases included in OmniPath. These supply large sets of directed interactions.

pypath.resources.data_formats.**interaction_htp = {'biogrid': <pypath.internals.input_format**

Transcriptional regulatory interactions.

pypath.resources.data_formats.**obsolate = {'nci_pid': <pypath.internals.input_formats.Netwo**

Reaction databases. These are not included in OmniPath, because only a minor part of their content can be used when processing along strict conditions to have only binary interactions with references.

pypath.resources.data_formats.**transcription_deprecated = {'oreganno_old': <pypath.internal**

miRNA-target resources

pypath.resources.data_formats.**transcription_onebyone = {'abs': <pypath.internals.input_fo**

New default transcription dataset is only TFregulons as it is already an integrated resource and has sufficient coverage.

import pypath

load only *A* confidence level: pypath.data_formats.transcription['tfregulons'].input_args['levels'] = {'A'} pa = pypath.PyPath() pa.init_network(pypath.data_formats.transcription)

pypath.data_formats.transcription['tfregulons'].input_args['levels'] = { 'A', 'B', 'C', 'D'

} pa = pypath.PyPath() pa.init_network(pypath.data_formats.transcription)

2.10 dataio

class pypath.inputs.main.**CellPhoneDBAnnotation**(*receptor*, *receptor_class*, *peripheral*, *secreted*, *secreted_class*, *transmembrane*, *integrin*)

integrin
Alias for field number 6

peripheral
Alias for field number 2

receptor
Alias for field number 0

receptor_class
Alias for field number 1

secreted
Alias for field number 3

secreted_class
Alias for field number 4

transmembrane
Alias for field number 5

class pypath.inputs.main.**ResidueMapper**
This class stores and serves the PDB –> UniProt residue level mapping. Attempts to download the mapping, and stores it for further use. Converts PDB residue numbers to the corresponding UniProt ones.

clean()
Removes cached mappings, freeing up memory.

pypath.inputs.main.**acsn_interactions**(*keep_in_complex_interactions=True*)
Processes ACSN data from local file. Returns list of interactions.

@keep_in_complex_interactions [bool] Whether to include interactions from complex expansion.

pypath.inputs.main.**biogrid_interactions**(*organism=9606*, *htp_limit=1*, *ltp=True*)
Downloads and processes BioGRID interactions. Keeps only the "low throughput" interactions. Returns list of interactions.

@organism [int] NCBI Taxonomy ID of organism.

@htp_limit [int] Exclude interactions only from references cited at more than this number of interactions.

pypath.inputs.main.**cancer_gene_census_annotations**(*user=None*, *passwd=None*, *credentials_fname='cosmic_credentials'*)
Retrieves a list of cancer driver genes (Cancer Gene Census) from the Sanger COSMIC (Catalogue of Somatic Mutations in Cancer) database.

Does not work at the moment (signature does not match error).

pypath.inputs.main.**cancersea_annotations**()
Retrieves genes annotated with cancer funcitonal states from the CancerSEA database.

pypath.inputs.main.**cellphonedb_ligands_receptors**()
Retrieves the set of ligands and receptors from CellPhoneDB. Returns tuple of sets.

pypath.inputs.main.**cellphonedb_protein_annotations**(*add_complex_annotations=True*)

Parameters **add_complex_annotations** (*bool*) – Copy the annotations of complexes to each of their member proteins.

pypath.inputs.main.**compleat_complexes**(*predicted=True*)
Retrieves complexes from the Compleat database.

pypath.inputs.main.**complexportal_complexes**(*organism=9606*, *return_details=False*)
Complex dataset from IntAct. See more: http://www.ebi.ac.uk/intact/complex/ http://nar.oxfordjournals.org/content/early/2014/10/13/nar.gku975.full.pdf

pypath.inputs.main.**comppi_interaction_locations**(*organism=9606*)
Downloads and preprocesses protein interaction and cellular compartment association data from the ComPPI database. This data provides scores for occurrence of protein-protein interactions in various compartments.

pypath.inputs.main.**cpad_pathway_cancer**()
Collects only the pathway-cancer relationships. Returns sets of records grouped in dicts by cancer and by pathway.

pypath.inputs.main.**dgidb_annotations**()
Downloads druggable protein annotations from DGIdb.

pypath.inputs.main.**dip_login**(*user*, *passwd*)
This does not work for unknown reasons.

In addition, the binary_data parameter of Curl().__init__() has been changed, below updates are necessary.

pypath.inputs.main.**disgenet_annotations**(*dataset='curated'*)
Downloads and processes the list of all human disease related proteins from DisGeNet. Returns dict of dicts.

@dataset [str] Name of DisGeNet dataset to be obtained: *curated*, *literature*, *befree* or *all*.

pypath.inputs.main.**dorothea_interactions**(*levels={'A', 'B'}*, *only_curated=False*)
Retrieves TF-target interactions from TF regulons.

Parameters

- **levels** (*set*) – Confidence levels to be used.
- **only_curated** (*bool*) – Retrieve only literature curated interactions.

TF regulons is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

pypath.inputs.main.**elm_interactions**()
Downlods manually curated interactions from ELM. This is the gold standard set of ELM.

pypath.inputs.main.**get_3dcomplex**()
Downloads and preprocesses data from the 3DComplex database.

Returns dict of dicts where top level keys are PDB IDs, second level keys are pairs of tuples of UniProt IDs and values are list with the number of amino acids in contact.

pypath.inputs.main.**get_acsn_effects**()
Processes ACSN data, returns list of effects.

pypath.inputs.main.**get_ca1**()
Downloads and processes the CA1 signaling network (Ma'ayan 2005). Returns list of interactions.

pypath.inputs.main.**get_ccmap**(*organism=9606*)
Downloads and processes CancerCellMap. Returns list of interactions.

@organism [int] NCBI Taxonomy ID to match column #7 in nodes file.

pypath.inputs.main.**get_csa**(*uniprots=None*)
Downloads and preprocesses catalytic sites data. This data tells which residues are involved in the catalytic activity of one protein.

pypath.inputs.main.**get_dgidb_old**()
Deprecated. Will be removed soon.

Downloads and processes the list of all human druggable proteins. Returns a list of GeneSymbols.

pypath.inputs.main.**get_domino_ptms**()
The table comes from dataio.get_domino(), having the following fields: header = ['uniprot-A', 'uniprot-B', 'isoform-A', 'isoform-B', #3 'exp. method', 'references', 'taxon-A', 'taxon-B', #7 'role-A', 'role-B', 'binding-site-range-A', 'binding-site-range-B', #11 'domains-A', 'domains-B', 'ptm-residue-A', 'ptm-residue-B', #15 'ptm-type-mi-A', 'ptm-type-mi-B', 'ptm-type-A', 'ptm-type-B', #19 'ptm-res-name-A', 'ptm-res-name-B', 'mutations-A', 'mutations-B', #23 'mutation-effects-A', 'mutation-effects-B', 'domains-interpro-A', #26 'domains-interpro-B', 'negative'] #28

pypath.inputs.main.**get_dorothea**(*levels={'A', 'B'}*, *only_curated=False*)
Retrieves TF-target interactions from TF regulons.

Parameters

- **levels** (*set*) – Confidence levels to be used.
- **only_curated** (*bool*) – Retrieve only literature curated interactions.

TF regulons is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

pypath.inputs.main.**get_exocarta**(*organism=9606*, *types=None*)

Parameters **types** (*set*) – Molecule types to retrieve. Possible values: *protein*, *mrna*.

pypath.inputs.main.**get_go_desc**(*go_ids*, *organism=9606*)
Deprecated, should be removed soon.

pypath.inputs.main.**get_go_quick**(*organism=9606*, *slim=False*, *names_only=False*, *aspects=('C', 'F', 'P')*)
Deprecated, should be removed soon.

Loads GO terms and annotations from QuickGO. Returns 2 dicts: *names* are GO terms by their IDs, *terms* are proteins GO IDs by UniProt IDs.

pypath.inputs.main.**get_graphviz_attrs**()
Downloads graphviz attribute list from graphviz.org. Returns 3 dicts of dicts: graph_attrs, vertex_attrs and edge_attrs.

pypath.inputs.main.**get_guide2pharma**(*organism='human'*, *endogenous=True*, *process_interactions=True*, *process_complexes=True*)
Downloads and processes Guide to Pharmacology data. Returns list of dicts.

@organism [str] Name of the organism, e.g. *human*.

@endogenous [bool] Whether to include only endogenous ligands interactions.

pypath.inputs.main.**get_havugimana**()
Downloads data from Supplement Table S3/1 from Havugimana 2012 Cell. 150(5): 1068–1081.

pypath.inputs.main.**get_hpmr**(*use_cache=None*)
Downloads ligand-receptor and receptor-receptor interactions from the Human Plasma Membrane Receptome database.

pypath.inputs.main.**get_hpmr_old**()
Deprecated, should be removed soon.

Downloads and processes the list of all human receptors from human receptor census (HPMR – Human Plasma Membrane Receptome). Returns list of GeneSymbols.

pypath.inputs.main.**get_hsn**()
Downloads and processes HumanSignalingNetwork version 6 (published 2014 Jan by Edwin Wang). Returns list of interactions.

pypath.inputs.main.**get_i3d**()
Interaction3D contains residue numbers in given chains in given PDB stuctures, so we need to add an offset to get the residue numbers valid for UniProt sequences. Offsets can be obtained from Instruct, or from the Pfam PDB-chain-UniProt mapping table.

pypath.inputs.main.**get_ielm**(*ppi*, *id_type='UniProtKB_AC'*, *mydomains='HMMS'*, *maxwait=180*, *cache=True*, *part=False*, *part_size=500*, *headers=None*)
Performs one query to iELM. Parameters are the same as at get_ielm_huge().

pypath.inputs.main.**get_ielm_huge**(*ppi*, *id_type='UniProtKB_AC'*, *mydomains='HMMS'*, *maxwait=180*, *cache=True*, *part_size=500*, *headers=None*)
Loads iELM predicted domain-motif interaction data for a set of protein-protein interactions. This method breaks the list into reasonable sized chunks and performs multiple requests to iELM, and also retries in case of failure, with reducing the request size. Provides feedback on the console.

Parameters

- **id_type** (*str*) – The type of the IDs in the supplied interaction list. Default is 'UniProtKB_AC'. Please refer to iELM what type of IDs it does understand.
- **mydomains** (*str*) – The type of the domain detection method. Defaults to 'HMMS'. Please refer to iELM for alternatives.
- **maxwait** (*int*) – The limit of the waiting time in seconds.
- **cache** (*bool*) – Whether to use the cache or download everything again.
- **part_size** (*int*) – The number of interactions to be queried in one request.
- **headers** (*list*) – Additional HTTP headers to send to iELM with each request.

pypath.inputs.main.**get_instruct**()
Instruct contains residue numbers in UniProt sequences, it means no further calculations of offsets in chains of PDB structures needed. Chains are not given, only a set of PDB structures supporting the domain-domain // protein-protein interaction.

pypath.inputs.main.**get_instruct_offsets**()
These offsets should be understood as from UniProt to PDB.

pypath.inputs.main.**get_integrins**()
Returns a set of the UniProt IDs of the human integrins from Table 1 of Takada et al 2007 (10.1186/gb-2007-8-5-215).

pypath.inputs.main.**get_laudanna_directions**()
Downloads and processes the SignalingFlow edge attributes from Laudanna Lab. Returns list of directions.

pypath.inputs.main.**get_laudanna_effects**()
Downloads and processes the SignalingDirection edge attributes from Laudanna Lab. Returns list of effects.

pypath.inputs.main.**get_listof_ontologies**()
Returns a list of available ontologies using the bioservices module.

pypath.inputs.main.**get_ontology**(*ontology*)
Downloads an ontology using the bioservices module.

pypath.inputs.main.**get_pepcyber**(*cache=None*)
Downloads phosphoprotein binding protein interactions from the PEPCyber database.

pypath.inputs.main.**get_pmid**(*idList*)
For a list of doi or PMC IDs fetches the corresponding PMIDs.

pypath.inputs.main.**get_switches_elm**()
switches.elm is a resource containing functional switches in molecular regulation, in domain-motif level resolution, classified into categories according to their mechanism.

pypath.inputs.main.**get_tfcensus**(*classes=('a', 'b', 'other')*)
Downloads and processes the list of all known transcription factors from TF census (Vaquerizas 2009). This resource is human only. Returns set of UniProt IDs.

pypath.inputs.main.**get_tfregulons**(*levels={'A', 'B'}*, *only_curated=False*)
Retrieves TF-target interactions from TF regulons.

Parameters

- **levels** (*set*) – Confidence levels to be used.
- **only_curated** (*bool*) – Retrieve only literature curated interactions.

TF regulons is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

pypath.inputs.main.**get_tfregulons_old**(*levels={'A', 'B'}*, *only_curated=False*)
Retrieves TF-target interactions from TF regulons.

Parameters

- **levels** (*set*) – Confidence levels to be used.
- **only_curated** (*bool*) – Retrieve only literature curated interactions.

TF regulons is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

pypath.inputs.main.**get_vesiclepedia**(*organism=9606*, *types=None*)

Parameters **types** (*set*) – Molecule types to retrieve. Possible values: *protein*, *mrna*.

pypath.inputs.main.**go_ancestors**(*aspects=('C', 'F', 'P')*)
Queries the ancestors of GO terms by QuickGO REST API.

Returns dict of sets where keys are GO accessions and values are sets of their ancestors.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_ancestors_goose**(*aspects=('C', 'F', 'P')*)
Queries the ancestors of GO terms by AmiGO goose.

Returns dict of sets where keys are GO accessions and values are sets of their ancestors.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_ancestors_quickgo**(*aspects=('C', 'F', 'P')*)
Queries the ancestors of GO terms by QuickGO REST API.

Returns dict of sets where keys are GO accessions and values are sets of their ancestors.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_annotations**(*organism='human'*)
Downloads GO annotation from UniProt GOA.

pypath.inputs.main.**go_annotations_goa**(*organism='human'*)
Downloads GO annotation from UniProt GOA.

pypath.inputs.main.**go_annotations_goose**(*organism=9606*, *aspects=('C'*, *'F'*, *'P')*, *uniprots=None*)
Queries GO annotations by AmiGO goose.

IMPORTANT: This is not the preferred method any more to get terms and annotations. Recently the preferred method to access GO annotations is `pypath.dataio.go_annotations_solr()`. The data in GO MySQL instances has not been updated since Dec 2016. Unfortunately the providers ceased to support MySQL, the most flexible and highest performance access to GO data. The replacement is Solr which is far from providing the same features as MySQL.

Returns terms in dict of dicts and annotations in dict of dicts of sets. In both dicts the keys are aspects by their one letter codes. In the term dicts keys are GO accessions and values are their names. In the annotation dicts keys are UniProt IDs and values are sets of GO accessions.

Parameters

- **organism** (*int*) – NCBI Taxonomy ID of one organism. Default is human (9606).
- **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.
- **uniprots** (*list*) – Optionally a list of UniProt IDs. If *None*, results for all proteins returned.

pypath.inputs.main.**go_annotations_quickgo**(*organism=9606*, *aspects=('C'*, *'F'*, *'P')*, *relations=('is_a'*, *'part_of')*)
Queries GO annotations by QuickGO REST API.

IMPORTANT: Recently the preferred method to access GO annotations is `pypath.dataio.go_annotations_goa()`. Contrary to its name QuickGO is super slow, otherwise it should yield up to date data, identical to the GOA file.

Returns terms in dict of dicts and annotations in dict of dicts of sets. In both dicts the keys are aspects by their one letter codes. In the term dicts keys are GO accessions and values are their names. In the annotation dicts keys are UniProt IDs and values are sets of GO accessions.

Parameters

- **organism** (*int*) – NCBI Taxonomy ID of one organism. Default is human (9606).
- **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.
- **uniprots** (*list*) – Optionally a list of UniProt IDs. If *None*, results for all proteins returned.

pypath.inputs.main.**go_annotations_solr**(*organism=9606*, *aspects=('C'*, *'F'*, *'P')*, *references=False*)
Queries GO annotations by AmiGO Solr.

Before other methods have been provided to access GO. Now this is the preferred method to get annotations. Returns terms in dict of dicts and annotations in dict of dicts of sets. In both dicts the keys are aspects by their one letter codes. In the term dicts keys are GO accessions and values are their names. In the annotation dicts keys are UniProt IDs and values are sets of GO accessions.

Parameters

- **organism** (*int*) – NCBI Taxonomy ID of one organism. Default is human (9606).
- **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.
- **references** (*bool*) – Retrieve the references (PubMed IDs) for the annotations. Currently not implemented.

pypath.inputs.main.**go_annotations_uniprot**(*organism=9606*, *swissprot='yes'*)
Deprecated, should be removed soon.

pypath.inputs.main.**go_descendants**(*aspects=('C', 'F', 'P')*, *terms=None*, *relations=None*, *quickgo_download_size=500*)
Queries descendants of GO terms by QuickGO REST API.

Returns dict of sets where keys are GO accessions and values are sets of their descendants.

Parameters

- **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.
- **terms** (*dict*) – Result from `go_terms_solr`. If `None` the method will be called.

pypath.inputs.main.**go_descendants_goose**(*aspects=('C', 'F', 'P')*)
Queries descendants of GO terms by AmiGO goose.

IMPORTANT: This is not the preferred method any more to get descendants. Recently the preferred method to access GO annotations is `pypath.dataio.go_descendants_quickgo()`. The data in GO MySQL instances has not been updated since Dec 2016. Unfortunately the providers ceased to support MySQL, the most flexible and highest performance access to GO data. The replacement is Solr which is far from providing the same features as MySQL, for example it is unable to provide GO graph relationships. Other service is QuickGO which is up to date and has nice ways to query the ontology.

Returns dict of sets where keys are GO accessions and values are sets of their descendants.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_descendants_quickgo**(*aspects=('C', 'F', 'P')*, *terms=None*, *relations=None*, *quickgo_download_size=500*)
Queries descendants of GO terms by QuickGO REST API.

Returns dict of sets where keys are GO accessions and values are sets of their descendants.

Parameters

- **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.
- **terms** (*dict*) – Result from `go_terms_solr`. If `None` the method will be called.

pypath.inputs.main.**go_descendants_to_ancestors**(*desc*)
Turns a dict of descendants to dict of ancestors by swapping the relationships. This way descendants will be the keys and their ancestors will be the values.

pypath.inputs.main.**go_terms**(*aspects=('C', 'F', 'P')*)
Queries GO terms by the QuickGO REST API.

Return dict of dicts where upper level keys are one letter codes of the aspects *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively. Lower level keys are GO accessions and values are names of the terms.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_terms_goose**(*aspects=('C', 'F', 'P')*)
Queries GO terms by AmiGO goose.

Return dict of dicts where upper level keys are one letter codes of the aspects *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively. Lower level keys are GO accessions and values are names of the terms.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_terms_quickgo**(*aspects=('C', 'F', 'P')*)
Queries GO terms by the QuickGO REST API.

Return dict of dicts where upper level keys are one letter codes of the aspects *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively. Lower level keys are GO accessions and values are names of the terms.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**go_terms_solr**(*aspects=('C', 'F', 'P')*)
Queries GO terms by AmiGO Solr.

Returns dict of dicts where upper level keys are one letter codes of the aspects *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively. Lower level keys are GO accessions and values are names of the terms.

Parameters **aspects** (*tuple*) – GO aspects: *C*, *F* and *P* for cellular_component, molecular_function and biological_process, respectively.

pypath.inputs.main.**havugimana_complexes**()
Retrieves complexes from Supplement Table S3/1 from Havugimana 2012 Cell. 150(5): 1068–1081.

pypath.inputs.main.**hpmr_interactions_old**()
Deprecated, should be removed soon.

Downloads ligand-receptor and receptor-receptor interactions from the Human Plasma Membrane Receptome database.

pypath.inputs.main.**intogen_annotations**()
Returns a list of cancer driver genes according to the IntOGen database.

pypath.inputs.main.**kegg_interactions**()
Downloads and processes KEGG Pathways. Returns list of interactions.

pypath.inputs.main.**kinasedotcom_annotations**()
Downloads and processes kinase annotations from kinase.com.

pypath.inputs.main.**lmpid_dmi**(*organism=9606*)
Converts list of domain-motif interactions supplied by *pypath.dataio.load_lmpid()* to list of ‘pypath.intera.DomainMotif() objects.

pypath.inputs.main.**lmpid_interactions**(*organism=9606*)
Converts list of domain-motif interactions supplied by *pypath.dataio.load_lmpid()* to list of interactions.

pypath.inputs.main.**load_lmpid**(*organism=9606*)
Reads and processes LMPID data from local file *pypath.data/LMPID_DATA_pubmed_ref.xml*. The file was provided by LMPID authors and is now redistributed with the module. Returns list of domain-motif interactions.

pypath.inputs.main.**load_macrophage**()
Loads Macrophage from local file. Returns list of interactions.

pypath.inputs.main.**matrisome_annotations**(*organism=9606*)
Downloads MatrisomeDB 2.0, a database of extracellular matrix proteins. Returns dict where keys are UniProt IDs and values are tuples of classes, subclasses and notes.

pypath.inputs.main.**matrixdb_ecm_proteins**(*organism=9606*)
Returns a set of ECM (extracellular matrix) protein UniProt IDs retrieved from MatrixDB.

pypath.inputs.main.**matrixdb_membrane_proteins**(*organism=9606*)
Returns a set of membrane protein UniProt IDs retrieved from MatrixDB.

pypath.inputs.main.**matrixdb_secreted_proteins**(*organism=9606*)
Returns a set of secreted protein UniProt IDs retrieved from MatrixDB.

pypath.inputs.main.**msigdb_annotations**(*registered_email=None*, *only_collections=None*, *exclude=('c5',)*)
Downloads all or some MSigDB gene set collections and processes them to an annotation type dictionary.

Parameters

- **registered_email** (*str,NoneType*) – An email address registered at MSigDB. If *None* the *msigdb_email* from `pypath.settings` will be used.
- **only_collections** (*set,NoneType*) – Limit the annotations only to these collections. For available collections e.g. `{'h.all', 'c2cgp'}` refer to the MSigDB webpage: http://software.broadinstitute.org/gsea/downloads.jsp#msigdb
- **exclude** (*tuple*) – Exclude the collections having their name starting with any of the strings in this tuple. By default *c5* (Gene Ontology) is excluded.

pypath.inputs.main.**msigdb_download**(*registered_email=None*, *collection='msigdb'*, *id_type='symbols'*, *force_download=False*)
Downloads and preprocesses a collection of gmt format gene sets from MSigDB. Returns dict of sets with gene set names as keys and molecular identifiers as values.

Parameters

- **registered_email** (*str,NoneType*) – An email address registered at MSigDB. If *None* the *msigdb_email* from `pypath.settings` will be used.
- **collection** (*str*) – The name of the gene set collection. For available collections (e.g. *h.all* or *c2.cpg*) refer to the MSigDB website: http://software.broadinstitute.org/gsea/downloads.jsp#msigdb The default value *msigdb* contains all the genesets however you won't be able to distinguish which geneset comes from which collection. For this you need to download the collections one by one.
- **id_type** (*str*) – MSigDB provides Gene Symbols (*symbols*) and Entrez Gene IDs (*entrez*).
- **force_download** (*bool*) – Download even if cache content is available.

pypath.inputs.main.**msigdb_download_collections**(*registered_email=None*, *only_collections=None*, *exclude=('c5',)*, *id_type='symbols'*)
Downloads all or some MSigDB gene set collections. Returns a dict of dicts where upper level keys are collections while lower level keys are geneset names and values are molecular identifiers.

Parameters

- **registered_email** (*str,NoneType*) – An email address registered at MSigDB. If *None* the *msigdb_email* from `pypath.settings` will be used.

- **only_collections** (*set,NoneType*) – Limit the annotations only to these collections. For available collections e.g. `{'h.all', 'c2cgp'}` refer to the MSigDB webpage: http://software.broadinstitute.org/gsea/downloads.jsp#msigdb
- **exclude** (*tuple*) – Exclude the collections having their name starting with any of the strings in this tuple. By default *c5* (Gene Ontology) is excluded.

pypath.inputs.main.**only_pmids**(*idList*, *strict=True*)
Return elements unchanged which comply with the PubMed ID format, and attempts to translate the DOIs and PMC IDs using NCBI E-utils. Returns list containing only PMIDs.

@idList [list, str] List of IDs or one single ID.

@strict [bool] Whether keep in the list those IDs which are not PMIDs, neither DOIs or PMC IDs or NIH manuscript IDs.

pypath.inputs.main.**open_pubmed**(*pmid*)
Opens PubMed record in web browser.

@pmid [str or int] PubMed ID

pypath.inputs.main.**pdzbase_interactions**()
Downloads data from PDZbase. Parses data from the HTML tables.

pypath.inputs.main.**phosphatome_annotations**()
Downloads the list of phosphatases from Chen et al, Science Signaling (2017) Table S1.

pypath.inputs.main.**ramilowski_interactions**(*putative=False*)
Downloads and processes ligand-receptor interactions from Supplementary Table 2 of Ramilowski 2015.

pypath.inputs.main.**reactions_biopax**(*biopax_file*, *organism=9606*, *protein_name_type='UniProt'*, *clean=True*)
Processes a BioPAX file and extracts binary interactions.

pypath.inputs.main.**reactome_biopax**(*organism=9606*, *cache=True*)
Downloads Reactome human reactions in SBML format. Returns File object.

pypath.inputs.main.**reactome_interactions**(*cacheFile=None*, ***kwargs*)
Downloads and processes Reactome BioPAX. Extracts binary interactions. The applied criteria are very stringent, yields very few interactions. Requires large free memory, approx. 2G.

pypath.inputs.main.**reactome_sbml**()
Downloads Reactome human reactions in SBML format. Returns gzip.GzipFile object.

pypath.inputs.main.**signalink_interactions**()
Reads and processes SignaLink3 interactions from local file. Returns list of interactions.

pypath.inputs.main.**surfaceome_annotations**()
Downloads the "In silico human surfaceome". Dict with UniProt IDs as key and tuples of surface prediction score, class and subclass as values (columns B, N, S and T of table S3).

pypath.inputs.main.**take_a_trip**(*cachefile=None*)
Downloads TRIP data from webpage and preprocesses it. Saves preprocessed data into *cachefile* and next time loads from this file.

Parameters **str** (*cachefile*) – Path to pickle dump of preprocessed TRIP database. If does not exist the database will be downloaded and saved to this file. By default the path queried from the `settings` module.

pypath.inputs.main.**tfregulons_interactions**(*levels={'A', 'B'}*, *only_curated=False*)
Retrieves TF-target interactions from TF regulons.

Parameters

- **levels** (*set*) – Confidence levels to be used.
- **only_curated** (*bool*) – Retrieve only literature curated interactions.

TF regulons is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

pypath.inputs.main.**trip_find_uniprot**(*soup*)
Looks up a UniProt name in table downloaded from TRIP webpage.

@soup [bs4.BeautifulSoup] The *BeautifulSoup* instance returned by *pypath.dataio.trip_get_uniprot()*.

pypath.inputs.main.**trip_get_uniprot**(*syn*)
Downloads table from TRIP webpage and UniProt attempts to look up the UniProt ID for one synonym.

@syn [str] The synonym as shown on TRIP webpage.

pypath.inputs.main.**trip_interactions**(*exclude_methods=['Inference', 'Speculation'], predictions=False, species='Human', strict=False*)
Obtains processed TRIP interactions by calling *pypath.dataio.trip_process()* and returns list of interactions. All arguments are passed to *trip_process()*, see their definition there.

pypath.inputs.main.**trip_process**(*exclude_methods=['Inference', 'Speculation'], predictions=False, species='Human', strict=False*)
Downloads TRIP data by calling *pypath.dadio.take_a_trip()* and further provcesses it. Returns dict of dict with TRIP data.

@exclude_methods [list] Interaction detection methods to be discarded.

@predictions [bool] Whether to include predicted interactions.

@species [str] Organism name, e.g. *Human*.

@strict [bool] Whether include interactions with species not used as a bait or not specified.

pypath.inputs.main.**trip_process_table**(*tab*, *result*, *intrs*, *trp_uniprot*)
Processes one HTML table downloaded from TRIP webpage.

@tab [bs4.element.Tag()] One table of interactions from TRIP webpage.

@result [dict] Dictionary the data should be filled in.

@intrs [dict] Dictionary of already converted interactor IDs. This serves as a cache so do not need to look up the same ID twice.

@trp_uniprot [str] UniProt ID of TRP domain containing protein.

pypath.inputs.main.**wang_interactions**()
Downloads and processes Wang Lab HumanSignalingNetwork. Returns list of interactions.

pypath.inputs.main.**zhong2015_annotations**()
From 10.1111/nyas.12776 (PMID 25988664).

2.11 descriptions

pypath.resources.descriptions.**gen_html**()
Generates a HTML page from the *descriptions* array. This HTML is provided by the webservice under */info*, or can be saved locally with *write_html()*.

pypath.resources.descriptions.**write_html**(*filename='resources.html'*)
Saves the HTML descriptions to custom local file.

2.12 entity

Provides classes for representing molecular entities and their collections. A molecular entity is defined by its identifier, type and taxon.

class pypath.core.entity.**Entity**(*identifier*, *entity_type='protein'*, *id_type='uniprot'*, *taxon=9606*, *attrs=None*)
Represents a molecular entity such as protein, miRNA, lncRNA or small molecule.

Parameters

- **identifier** (*str*) – An identifier from the reference database e.g. UniProt ID for proteins.
- **entity_type** (*str*) – The type of the molecular entity, defaults to `'protein'`.
- **id_type** (*str*) – The type of the identifier (the reference database), default is `'uniprot'`.
- **taxon** (*int*) – The NCBI Taxonomy Identifier of the molecular entity, e.g. `9606` for human. Use `0` for non taxon specific molecules e.g. metabolites or drug compounds.
- **attrs** (*NoneType,dict*) – A dictionary of additional attributes.

class pypath.core.entity.**EntityKey**(*identifier*, *id_type*, *entity_type*, *taxon*)

entity_type
Alias for field number 2

id_type
Alias for field number 1

identifier
Alias for field number 0

taxon
Alias for field number 3

2.13 evidence

Provides classes for representing and processing evidences supporting relationships. The evidences hold information about the databases and literature references, they can be organized into collections. A number of operations are available on evidences and their collections, for example they can be combined or filtered.

class pypath.core.evidence.**Evidence**(*resource*, *references=None*)
Represents an evidence supporting a relationship such as molecular interaction, molecular complex, enzyme-PTM interaction, annotation, etc.

The evidence consists of two main parts: the database and the literature references. If a relationship is supprted by multiple databases, for each one *Evidence* object should be created and

Parameters

- **resource** (*pypath.resource.ResourceAttributes*) – An object derived from `pypath.resource.ResourceAttributes`.

- **references** (*str,list,set,NoneType*) – Optional, one or more literature references (preferably PubMed IDs).

has_interaction_type(*interaction_type*, *database=None*, *via=False*)
If `via` is `False` then it will be ignored, otherwise if `None` only primary resources are considered.

merge(*other*)
Merges two evidences. Returns set of either one or two evidences depending on whether the two evidences are from the same resource.

reload()
Reloads the object from the module level.

class pypath.core.evidence.**Evidences**(*evidences=()*)
A collection of evidences. All evidences supporting a relationship such as molecular interaction, molecular complex, enzyme-PTM interaction, annotation, etc should be collected in one *Evidences* object. This way the set of evidences can be queried a comprehensive way.

Parameters **evidences** (*tuple,list,set,Evidences*) – An iterable providing *Evidence* instances. It is possible to create an empty evidence collection and populate it later or to show this way that certain relationship has no supporting evidences.

has_interaction_type(*interaction_type*, *database=None*, *via=False*)
If `via` is `False` then it will be ignored, otherwise if `None` only primary resources are considered.

reload()
Reloads the object from the module level.

2.14 export

2.15 go

pypath.utils.go.**annotate**(*graph*, *organism=9606*, *aspects=('C', 'F', 'P')*)
Adds Gene Ontology annotations to the nodes of a graph.

Parameters **graph** (*igraph.Graph*) – Any `igraph.Graph` object with uniprot IDs in its `name` vertex attribute.

pypath.utils.go.**get_db**(*organism=9606*, *pickle_file=None*, *use_pickle_cache=True*)
Retrieves the current database instance and initializes it if does not exist yet.

pypath.utils.go.**init_db**(*organism=9606*, *pickle_file=None*, *use_pickle_cache=True*)
Initializes or reloads the GO annotation database. The database will be assigned to the `db` attribute of this module.

pypath.utils.go.**load_go**(*graph*, *organism=9606*, *aspects=('C', 'F', 'P')*)
Adds Gene Ontology annotations to the nodes of a graph.

Parameters **graph** (*igraph.Graph*) – Any `igraph.Graph` object with uniprot IDs in its `name` vertex attribute.

2.16 homology

pypath.utils.homology.**get_homologene**()
Downloads the recent release of the NCBI HomoloGene database. Returns file pointer.

pypath.utils.homology.**homologene_dict**(*source*, *target*, *id_type*)
Returns orthology translation table as dict, obtained from NCBI HomoloGene data.

Parameters

- **source** (*int*) – NCBI Taxonomy ID of the source species (keys).
- **target** (*int*) – NCBI Taxonomy ID of the target species (values).
- **id_type** (*str*) – ID type to be used in the dict. Possible values: 'RefSeq', 'Entrez', 'GI', 'GeneSymbol'.

pypath.utils.homology.**homologene_uniprot_dict**(*source*, *target*, *only_swissprot=True*)
Returns orthology translation table as dict from UniProt to Uniprot, obtained from NCBI HomoloGene data. Uses RefSeq and Entrez IDs for translation.

Parameters

- **source** (*int*) – NCBI Taxonomy ID of the source species (keys).
- **target** (*int*) – NCBI Taxonomy ID of the target species (values).
- **only_swissprot** (*bool*) – Translate only SwissProt IDs.

2.17 input_formats

2.18 interaction

Here we define one class, the `Interaction` which provides a rich API for representing and querying molecular interactions. The interactions serve as the building elements of the network and the `pypath.network.Network` object largely relies on methods of the `Interaction` objects.

class pypath.core.interaction.**Interaction**(*a*, *b*, *id_type_a='uniprot'*, *id_type_b='uniprot'*, *entity_type_a='protein'*, *entity_type_b='protein'*, *taxon_a=9606*, *taxon_b=9606*)
Represents a unique pair of molecular entities interacting with each other. One *Interaction* object might represent multiple interactions i.e. with different direction or effect or type (e.g. transcriptional regulation and post-translational regulation), each supported by different evidences.

Parameters

- **a,b** (*str,pypath.entity.Entity*) – The two interacting partners. If an `pypath.entity.Entity` objects provided the other attributes (entity_type, id_type, taxon) will be ignored.
- **id_type_a,id_type_b** (*str*) – The identifier types for partner `a` and `b` e.g. `'uniprot'`.
- **entity_type_a,entity_type_b** (*str*) – The types of the molecular entities `a` and `b` e.g. `'protein'`.
- **taxon_a,taxon_b** (*int*) – The NCBI Taxonomy Identifiers of partner `a` and `b` e.g. `9606` for human.

Details The arguments `a` and `b` will be assigned to the attribute `a` and `b` in an alphabetical order, hence it's possible that argument `a` becomes attribute `b`.

add_evidence(*evidence*, *direction='undirected'*, *effect=0*, *references=None*)
Adds directionality information with the corresponding data source named. Modifies self attributes `dirs` and `sources`.

Parameters

- **evidence** (*resource.NetworkResource*, evidence.Evidence) – Either a `pypath.evidence.Evidence` object or a resource as `pypath.resource.NetworkResource` object. In the latter case the references can be provided in a separate argument.
- **direction** (*tuple*) – Or [str], the directionality key for which the value on `dirs` has to be set `True`.
- **effect** (*int*) – The causal effect of the interaction. 1 or 'stimulation' corresponds to a stimulatory, -1 or 'inhibition' to an inhibitory while 0 to an unknown or neutral effect.
- **references** (*set*, *NoneType*) – A set of references, used only if the resource have been provided as `NetworkResource` object.

add_sign(*direction*, *sign*, *resource=None*, *resource_name=None*, *interaction_type='PPI'*, *data_model=None*, ***kwargs*)
Sets sign and source information on a given direction of the edge. Modifies the attributes `positive` and `positive_sources` or `negative` and `negative_sources` depending on the sign. Direction is also updated accordingly, which also modifies the attributes `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Pair of edge nodes specifying the direction from which the information is to be set/updated.
- **sign** (*str*) – Specifies the type of interaction. Either `'positive'` or `'negative'`.
- **resource** (*set*) – Contains the name(s) of the source(s) from which the information was obtained.
- ****kwargs** – Passed to `pypath.resource.NetworkResource` if `resource` is not already a `NetworkResource` or `Evidence` instance.

complex_identifiers_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_identifiers_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_identifiers_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_identifiers_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_identifiers_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_identifiers_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_labels_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_labels_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_labels_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_labels_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_labels_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complex_labels_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complexes_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complexes_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complexes_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complexes_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complexes_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

complexes_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

consensus(*only_interaction_type=None*, *only_primary=False*, *by_references=False*, *by_reference_resource_pairs=True*)

Infers the consensus edge(s) according to the number of supporting sources. This includes direction and sign.

Returns (*list*) – Contains the consensus edge(s) along with the consensus sign. If there is no major directionality, both are returned. The structure is as follows: `['<source>', '<target>', '<(un)directed>', '<sign>']`

consensus_edges(*only_interaction_type=None*, *only_primary=False*, *by_references=False*, *by_reference_resource_pairs=True*)

Infers the consensus edge(s) according to the number of supporting sources. This includes direction and sign.

Returns (*list*) – Contains the consensus edge(s) along with the consensus sign. If there is no major directionality, both are returned. The structure is as follows: `['<source>', '<target>', '<(un)directed>', '<sign>']`

count_complex_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_complex_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_complexes(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities

in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_data_models(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves data models matching the criteria.

count_degrees_directed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_directed_in(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_directed_out(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_negative(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_negative_in(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_negative_out(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_non_directed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_positive(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_positive_in(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_positive_out(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_signed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_signed_in(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_signed_out(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_degrees_undirected(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

count_entities(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_interaction_types(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

count_interactions(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.

- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

count_interactions_0(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Returns unique interacting pairs without being aware of the direction.

count_interactions_directed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
**kwargs: see the docs of method `get_interactions`.

count_interactions_mutual(***kwargs*)
Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

count_interactions_negative(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
**kwargs: see the docs of method `get_interactions`.

count_interactions_non_directed(***kwargs*)
Returns *True* if any resource annotates this interaction without and no resource with direction.

**kwargs: see the docs of method `get_interactions`.

count_interactions_positive(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
**kwargs: see the docs of method `get_interactions`.

count_interactions_signed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
**kwargs: see the docs of method `get_interactions`.

count_interactions_undirected(***kwargs*)
Returns *True* if any resource annotates this interaction without direction.

**kwargs: see the docs of method `get_interactions`.

count_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_lncrna_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_lncrna_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_lncrnas(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_mirna_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_mirna_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_mirnas(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_protein_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_protein_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_proteins(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_references(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves references matching the criteria.

count_resource_names(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves resource names matching the criteria.

count_resource_names_via(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

count_resources(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

count_resources_via(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

count_small_molecule_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_small_molecule_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

count_small_molecules(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

data_models_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves data models matching the criteria.

data_models_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves data models matching the criteria.

data_models_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves data models matching the criteria.

data_models_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves data models matching the criteria.

data_models_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves data models matching the criteria.

data_models_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves data models matching the criteria.

degrees_directed_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_in_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_in_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_in_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_in_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_in_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B,

but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_in_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_out_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_out_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_out_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the

direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_out_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_out_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_directed_out_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the

direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_in_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_in_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_in_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_in_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_in_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_in_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_out_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_out_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_out_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_out_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_out_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_negative_out_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_non_directed_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_non_directed_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_non_directed_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_non_directed_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_non_directed_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_non_directed_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the

direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_in_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_in_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_in_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_in_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_in_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_in_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_out_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_out_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_out_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_out_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_out_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_positive_out_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_in_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_in_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the

query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_in_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_in_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_in_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_in_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_out_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_out_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_out_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_out_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_out_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_signed_out_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_undirected_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_undirected_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_undirected_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_undirected_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_undirected_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are

'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

degrees_undirected_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are 'IN', *'OUT'* and 'ALL' for incoming, outgoing and all connections, respectively. If the direction is False the only possible mode is ALL. If the direction is None and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

entities_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:pypath.entity.Entity objects, alternatives are labels identifiers.

entities_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:pypath.entity.Entity objects, alternatives are labels identifiers.

entities_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:pypath.entity.Entity objects, alternatives are labels identifiers.

entities_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

entities_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

entities_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

evaluate_evidences(*this_direction*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Selects the evidence collections matching the direction and effect criteria and then evaluates if any of the evidences in these collections match the evidence criteria.

generate_df_records(*by_source=False*, *with_references=False*)

Yields interaction records. It is a generator because one edge can be represented by one or more records depending on the signs and directions and other parameters

Parameters

- **by_source** (*bool*) – Yield separate records by resources. This way the node pairs will be redundant and you need to group later if you want unique interacting pairs. By default is `False` because for most applications unique interactions are preferred. If `False` the *refrences* field will still be present but with `None` values.

- **with_references** (*bool*) – Include the literature references. By default is `False` because you rarely need these and they increase the data size significantly.

get_complex_identifiers(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_complex_labels(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_complexes(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_data_models(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves data models matching the criteria.

get_degrees(*mode*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_directed(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_directed_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_directed_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_negative(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_negative_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_negative_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_non_directed(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_positive(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_positive_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_positive_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and

also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_signed(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_signed_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_signed_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_degrees_undirected(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns a *set* of nodes with the connections matching the direction, effect and evidence criteria. E.g. if the query concerns the incoming degrees with positive effect and the matching evidences show A activates B, but not the other way around, only "B" will be returned.

Parameters **mode** (*str*) – The type of degrees to be considered. Three possible values are `'IN'`, *'OUT'* and `'ALL'` for incoming, outgoing and all connections, respectively. If the `direction` is `False` the only possible mode is `ALL`. If the `direction` is `None` and also directed evidence(s) match the criteria these will overwrite the undirected evidences and only the directed result will be returned.

get_direction(*direction*, *resources=False*, *evidences=False*, *sources=False*, *resource_names=False*)

Returns the state (or *resources* if specified) of the given *direction*.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`). Pair of nodes from which direction information is to be retrieved.

- **resources** (*bool*) – Optional, `'False'` by default. Specifies if the `resources` information of the given direction is to be retrieved instead.

Returns (*bool* or *set*) – (if `resources=True`). Presence/absence of the requested direction (or the list of resources if specified). Returns `None` if *direction* is not valid.

get_directions(*src*, *tgt*, *resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)

Returns all directions with boolean values or list of sources.

Parameters

- **src** (*str*) – Source node.
- **tgt** (*str*) – Target node.
- **resources** (*bool*) – Optional, `False` by default. Specifies whether to return the `resources` attribute instead of `dirs`.

Returns Contains the `dirs` (or `resources` if specified) of the given edge.

get_entities(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_identifiers(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_interaction_types(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *entity_type=None*, *source_entity_type=None*, *target_entity_type=None*)

Retrieves interaction types matching the criteria.

get_interactions(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *entity_type=None*, *source_entity_type=None*, *target_entity_type=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

get_interactions_0(***kwargs*)
Returns unique interacting pairs without being aware of the direction.

get_interactions_directed(***kwargs*)
**kwargs: see the docs of method `get_interactions`.

get_interactions_mutual(***kwargs*)
Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

get_interactions_negative(***kwargs*)
**kwargs: see the docs of method `get_interactions`.

get_interactions_non_directed(***kwargs*)
Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

**kwargs: see the docs of method `get_interactions`.

get_interactions_non_directed_0(***kwargs*)
Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

get_interactions_positive(***kwargs*)
**kwargs: see the docs of method `get_interactions`.

get_interactions_signed(***kwargs*)
**kwargs: see the docs of method `get_interactions`.

get_interactions_undirected(***kwargs*)
Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

**kwargs: see the docs of method `get_interactions`.

get_interactions_undirected_0(***kwargs*)
Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

get_labels(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_lncrna_identifiers(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_lncrna_labels(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_lncrnas(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_mirna_identifiers(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_mirna_labels(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_mirnas(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_protein_identifiers(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.

- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_protein_labels(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_proteins(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_references(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves references matching the criteria.

get_resource_names(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves resource names matching the criteria.

get_resource_names_via(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves resource names via matching the criteria.

get_resources(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves resources matching the criteria.

get_resources_via(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves resources via matching the criteria.

get_sign(*direction*, *sign=None*, *evidences=False*, *resources=False*, *resource_names=False*, *sources=False*)

Retrieves the sign information of the edge in the given diretion. If specified in *sign*, only that sign's information will be retrieved. If specified in *sources*, the sources of that information will be retrieved instead.

Parameters

- **direction** (*tuple*) – Contains the pair of nodes specifying the directionality of the edge from which th information is to be retrieved.

- **sign** (*str*) – Optional, `None` by default. Denotes whether to retrieve the `'positive'` or `'negative'` specific information.
- **resources** (*bool*) – Optional, `False` by default. Specifies whether to return the resources instead of sign.

Returns (*list*) – If `sign=None` containing [bool] values denoting the presence of positive and negative sign on that direction, if `sources=True` the [set] of sources for each of them will be returned instead. If *sign* is specified, returns [bool] or [set] (if `sources=True`) of that specific direction and sign.

get_small_molecule_identifiers(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_small_molecule_labels(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

get_small_molecules(*entity_type=None*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*, *return_type=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

has_sign(*direction=None*, *resources=None*)

Checks whether the edge (or for a specific *direction*) has any signed information (about positive/negative interactions).

Parameters **direction** (*tuple*) – Optional, None by default. If specified, only the information of that direction is checked for sign.

Returns

(*bool*) – True if there exist any information on the sign of the interaction, False otherwise.

identifiers_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:pypath.entity.Entity objects, alternatives are labels identifiers.

identifiers_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:pypath.entity.Entity objects, alternatives are labels identifiers.

identifiers_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:pypath.entity.Entity objects, alternatives are labels identifiers.

identifiers_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities

in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

identifiers_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

identifiers_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

interaction_types_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

interaction_types_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

interaction_types_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

interaction_types_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

interaction_types_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

interaction_types_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves interaction types matching the criteria.

interactions_0_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns unique interacting pairs without being aware of the direction.

interactions_0_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns unique interacting pairs without being aware of the direction.

interactions_0_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns unique interacting pairs without being aware of the direction.

interactions_0_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns unique interacting pairs without being aware of the direction.

interactions_0_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns unique interacting pairs without being aware of the direction.

interactions_0_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns unique interacting pairs without being aware of the direction.

interactions_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

interactions_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.

- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

interactions_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

interactions_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions

will be returned. However the `count_interactions_undirected` method will return *1* in this case.

- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

interactions_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

interactions_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Returns one or two tuples of the interacting partners: one if only one direction, two if both directions match the query criteria. The tuple will be empty if no evidence matches the criteria.

Parameters

- **direction** (*NontType*, *bool*, *tuple*) – If *None* both undirected and directed, if *True* only directed, if a *tuple* of entities only the interactions with that specific direction will be considered. Unless you set this parameter to *True* this method will return both directions if one or more undirected resources present. If *False*, only the undirected interactions will be considered, and if any resource annotates this interaction as undirected both directions will be returned. However the `count_interactions_undirected` method will return *1* in this case.
- **effect** (*NoneType*, *bool*, *str*) – If *None* also interactions without effect, if *True* only the ones with any effect, if a string naming an effect only the interactions with that specific effect will be considered.
- **resources** (*NontType*, *str*, *set*) – Optionally limit the query to one or more resources.
- **data_model** (*NontType*, *str*, *set*) – Optionally limit the query to one or more data models e.g. *activity_flow*.
- **interaction_type** (*NontType*, *str*, *set*) – Optionally limit the query to one or more interaction types e.g. *PPI*.
- **via** (*NontType*, *bool*, *str*, *set*) – Optionally limit the query to certain secondary databases or if *False* consider only data from primary databases.
- **entity_type** (*str*) – Molecule type for both of the entities.
- **source_entity_type** (*str*) – Molecule type for the source entity.
- **target_entity_type** (*str*) – Molecule type for the target entity.

interactions_directed_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_directed_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_directed_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_directed_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_directed_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_directed_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_mutual_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

interactions_mutual_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

interactions_mutual_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

interactions_mutual_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

interactions_mutual_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

interactions_mutual_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

interactions_negative_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_negative_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_negative_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_negative_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_negative_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_negative_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_0_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_0_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_0_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_0_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_0_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_0_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected and none as directed, the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

**kwargs: see the docs of method `get_interactions`.

interactions_non_directed_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_non_directed_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_non_directed_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_non_directed_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, but only if no resource provide direction. However the `count_interactions_non_directed` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_positive_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

****kwargs**: see the docs of method `get_interactions`.

interactions_positive_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_positive_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_positive_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_positive_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_positive_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_signed_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_signed_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_signed_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_signed_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_signed_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_signed_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_0_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_0_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_0_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_0_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_0_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_0_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected the interacting pair as a sorted tuple will be returned inside a one element tuple.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

**kwargs: see the docs of method `get_interactions`.

interactions_undirected_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undi-

rected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_undirected_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_undirected_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

interactions_undirected_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Only the undirected interactions will be considered, if any resource annotates this interaction as undirected both directions will be returned, no matter if certain resources provide direction. However the `count_interactions_undirected` method will return *1* in this case.

****kwargs**: see the docs of method `get_interactions`.

is_directed()

Checks if edge has any directionality information.

Returns (*bool*) – Returns `True` if any of the `dirs` attribute values is `True` (except `'undirected'`), `False` otherwise.

is_directed_by_resources(*resources=None*)

Checks if edge has any directionality information from some resource(s).

Returns (*bool*) – Returns `True` if any of the `dirs` attribute values is `True` (except `'undirected'`), `False` otherwise.

is_inhibition(*direction=None*, *resources=None*)

Checks if any (or for a specific *direction*) interaction is inhibition (negative interaction).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, checks the `negative` attribute of that specific directionality. If not specified, checks both.

Returns (*bool*) – `True` if any interaction (or the specified *direction*) is inhibitory (negative).

is_loop()

Returns

`True` if the interaction is a loop edge i.e. its endpoints are the same node.

is_mutual(***kwargs*)
Note: undirected interactions does not count as mutual but only interactions with explicit direction information for both directions.

**kwargs: see the docs of method `get_interactions`.

is_mutual_by_resources(*resources=None*)
Checks if the edge has mutual directions (both A–>B and B–>A) according to some resource(s).

is_stimulation(*direction=None*, *resources=None*)
Checks if any (or for a specific *direction*) interaction is activation (positive interaction).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, checks the `positive` attribute of that specific directionality. If not specified, checks both.

Returns (*bool*) – `True` if any interaction (or the specified *direction*) is activatory (positive).

iter_evidences(*this_direction*, *direction=None*, *effect=None*)
Selects and yields evidence collections matching the direction and effect criteria.

iter_match_evidences(*this_direction*, *direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Selects the evidence collections matching the direction and effect criteria and yields collections matching the evidence criteria.

labels_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

labels_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

labels_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

labels_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

labels_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

labels_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_identifiers_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_identifiers_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_identifiers_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_identifiers_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.

- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_identifiers_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_identifiers_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_labels_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_labels_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_labels_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_labels_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_labels_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrna_labels_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrnas_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrnas_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrnas_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrnas_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrnas_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

lncrnas_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

majority_dir(*only_interaction_type=None*, *only_primary=True*, *by_references=False*, *by_reference_resource_pairs=True*)

Infers which is the major directionality of the edge by number of supporting sources.

Returns (*tuple*) – Contains the pair of nodes denoting the consensus directionality. If the number of sources on both directions is equal, `None` is returned. If there is no directionality information, `'undirected'` will be returned.

majority_sign(*only_interaction_type=None*, *only_primary=True*, *by_references=False*, *by_reference_resource_pairs=True*)

Infers which is the major sign (activation/inhibition) of the edge by number of supporting sources on both directions.

Returns (*dict*) – Keys are the node tuples on both directions (`straight`/`reverse`) and values can be either `None` if that direction has no sign information or a list of two [bool] elements corresponding to majority of positive and majority of negative support. In case both elements of the list are `True`, this means the number of supporting sources for both signs in that direction is equal.

merge(*other*)

Merges current Interaction with another (if and only if they are the same class and contain the same nodes). Updates the attributes `direction`, `positive` and `negative`.

Parameters **other** (*pypath.interaction.Interaction*) – The new Interaction object to be merged with the current one.

mirna_identifiers_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_identifiers_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_identifiers_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_identifiers_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_identifiers_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_identifiers_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_labels_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_labels_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_labels_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_labels_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_labels_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirna_labels_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirnas_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirnas_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirnas_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirnas_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirnas_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

mirnas_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

negative_a_b()

Checks if the `a_b` directionality is a negative interaction.

Returns (*bool*) – `True` if there is supporting information on the `a_b` direction of the edge as inhibition. `False` otherwise.

negative_b_a()

Checks if the `b_a` directionality is a negative interaction.

Returns (*bool*) – `True` if there is supporting information on the `b_a` direction of the edge as inhibition. `False` otherwise.

negative_resources_a_b(***kwargs*)

Retrieves the list of resources for the `a_b` direction and negative sign.

Returns (*set*) – Contains the names of the resources supporting the `a_b` directionality of the edge with a negative sign.

negative_resources_b_a(***kwargs*)

Retrieves the list of resources for the `b_a` direction and negative sign.

Returns (*set*) – Contains the names of the resources supporting the `b_a` directionality of the edge with a negative sign.

negative_reverse()

Checks if the `b_a` directionality is a negative interaction.

Returns (*bool*) – `True` if there is supporting information on the `b_a` direction of the edge as inhibition. `False` otherwise.

negative_straight()

Checks if the `a_b` directionality is a negative interaction.

Returns (*bool*) – `True` if there is supporting information on the `a_b` direction of the edge as inhibition. `False` otherwise.

positive_a_b()

Checks if the `a_b` directionality is a positive interaction.

Returns (*bool*) – `True` if there is supporting information on the `a_b` direction of the edge as activation. `False` otherwise.

positive_b_a()

Checks if the `b_a` directionality is a positive interaction.

Returns (*bool*) – `True` if there is supporting information on the `b_a` direction of the edge as activation. `False` otherwise.

positive_resources_a_b(***kwargs*)
Retrieves the list of resources for the `a_b` direction and positive sign.

Returns (*set*) – Contains the names of the resources supporting the `a_b` directionality of the edge with a positive sign.

positive_resources_b_a(***kwargs*)
Retrieves the list of resources for the `b_a` direction and positive sign.

Returns (*set*) – Contains the names of the resources supporting the `b_a` directionality of the edge with a positive sign.

positive_reverse()
Checks if the `b_a` directionality is a positive interaction.

Returns (*bool*) – `True` if there is supporting information on the `b_a` direction of the edge as activation. `False` otherwise.

positive_straight()
Checks if the `a_b` directionality is a positive interaction.

Returns (*bool*) – `True` if there is supporting information on the `a_b` direction of the edge as activation. `False` otherwise.

protein_identifiers_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_identifiers_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_identifiers_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities

in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_identifiers_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_identifiers_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_identifiers_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_labels_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_labels_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_labels_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_labels_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.

- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_labels_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

protein_labels_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

proteins_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

proteins_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

proteins_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

proteins_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

proteins_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

proteins_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

references_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves references matching the criteria.

references_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves references matching the criteria.

references_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves references matching the criteria.

references_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves references matching the criteria.

references_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves references matching the criteria.

references_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves references matching the criteria.

reload()
Reloads the object from the module level.

resource_names_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names matching the criteria.

resource_names_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names matching the criteria.

resource_names_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names matching the criteria.

resource_names_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names matching the criteria.

resource_names_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names matching the criteria.

resource_names_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names matching the criteria.

resource_names_via_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

resource_names_via_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

resource_names_via_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

resource_names_via_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

resource_names_via_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

resource_names_via_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resource names via matching the criteria.

resources_a_b(*resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)
Retrieves the list of resources for the `a_b` direction.

Returns (*set*) – Contains the names of the sources supporting the `a_b` directionality of the edge.

resources_b_a(*resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)
Retrieves the list of sources for the `b_a` direction.

Returns (*set*) – Contains the names of the sources supporting the `b_a` directionality of the edge.

resources_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

resources_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

resources_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

resources_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

resources_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

resources_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources matching the criteria.

resources_undirected(*resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)
Retrieves the list of resources without directed information.

Returns (*set*) – Contains the names of the sources supporting the edge presence but without specific directionality information.

resources_via_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

resources_via_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

resources_via_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

resources_via_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

resources_via_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

resources_via_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Retrieves resources via matching the criteria.

small_molecule_identifiers_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_identifiers_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_identifiers_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_identifiers_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=No*
interaction_type=None, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_identifiers_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_identifiers_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_labels_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_labels_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_labels_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_labels_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_labels_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecule_labels_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecules_by_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecules_by_interaction_type(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecules_by_interaction_type_and_data_model(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecules_by_interaction_type_and_data_model_and_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecules_by_reference(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.
- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

small_molecules_by_resource(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Retrieves the entities involved in interactions matching the criteria. It either returns both interacting entities in a *set* or an empty *set*. This may not sound so useful at the level of this object but becomes more useful once we want to collect entities having certain kind of interactions across a series of *Interaction* objects.

Parameters

- **entity_type** (*str*) – The type of the molecular entity. Possible values: *protein*, *complex*, *mirna*, *small_molecule*.

- **return_type** (*str*) – The type of values to return. Default is py:class:`pypath.entity.Entity` objects, alternatives are `labels` `identifiers`.

source(*undirected=False*, *resources=None*, ***kwargs*)
Returns the name(s) of the source node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the source node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

sources_reverse(*resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)
Retrieves the list of sources for the `b_a` direction.

Returns (*set*) – Contains the names of the sources supporting the `b_a` directionality of the edge.

sources_straight(*resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)
Retrieves the list of resources for the `a_b` direction.

Returns (*set*) – Contains the names of the sources supporting the `a_b` directionality of the edge.

sources_undirected(*resources=False*, *evidences=False*, *resource_names=False*, *sources=False*)
Retrieves the list of resources without directed information.

Returns (*set*) – Contains the names of the sources supporting the edge presence but without specific directionality information.

src(*undirected=False*, *resources=None*, ***kwargs*)
Returns the name(s) of the source node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the source node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

src_by_resource(*resource*)
Returns the name(s) of the source node(s) for each existing direction on the interaction for a specific *resource*.

Parameters **resource** (*str*) – Name of the resource according to which the information is to be retrieved.

Returns (*list*) – Contains the name(s) for the source node(s) according to the specified *resource*. This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the specified *source* is not found or invalid, an empty list will be returned.

target(*undirected=False*, *resources=None*, ***kwargs*)
Returns the name(s) of the target node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the target node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

tgt(*undirected=False*, *resources=None*, ***kwargs*)
Returns the name(s) of the target node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the target node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

tgt_by_resource(*resource*)
Returns the name(s) of the target node(s) for each existing direction on the interaction for a specific *resource*.

Parameters **resource** (*str*) – Name of the resource according to which the information is to be retrieved.

Returns (*list*) – Contains the name(s) for the target node(s) according to the specified *resource*. This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the specified *source* is not found or invalid, an empty list will be returned.

translate(*ids*, *new_attrs=None*)
Translates the node names/identifiers according to the dictionary *ids*. Also is able to change attributes like *id_type*, *taxon* and *entity_type*.

Parameters

- **ids** (*dict*) – Dictionary containing (at least) the current names of the nodes as keys and their translation as values.
- **new_attrs** (*dict*) – Dictionary with new IDs as keys and their dicts of their new attributes as values. For any attribute not provided here the attributes from the original instance will be used. E.g. you can provide '{'1956': {'id_type': 'entrez'}}' if the new ID type for protein EGFR is Entrez Gene ID.

Returns (*pypath.main.Direction*) – The copy of current edge object with translated node names.

unset_dir(*direction*, *only_sign=False*, *resource=None*, *interaction_type=None*, *via=False*, *source=None*)
Removes directionality and/or source information of the specified *direction*. Modifies attribute `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`) the pair of nodes specifying the directionality from which the information is to be removed.
- **resource** (*set*) – Optional, `None` by default. If specified, determines which specific source(s) is(are) to be removed from `sources` attribute in the specified *direction*.

unset_direction(*direction*, *only_sign=False*, *resource=None*, *interaction_type=None*, *via=False*, *source=None*)
Removes directionality and/or source information of the specified *direction*. Modifies attribute `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`) the pair of nodes specifying the directionality from which the information is to be removed.
- **resource** (*set*) – Optional, `None` by default. If specified, determines which specific source(s) is(are) to be removed from `sources` attribute in the specified *direction*.

unset_interaction_type(*interaction_type*)
Removes all evidences with a certain `interaction_type`.

unset_sign(*direction*, *sign*, *resource=None*, *interaction_type=None*, *via=False*, *source=None*)
Removes sign and/or source information of the specified *direction* and *sign*. Modifies at-

tribute `positive` and `positive_sources` or `negative` and `negative_sources` (or `positive_attributes`/`negative_sources` only if `source=True`).

Parameters

- **direction** (*tuple*) – The pair of nodes specifying the directionality from which the information is to be removed.
- **sign** (*str*) – Sign from which the information is to be removed. Must be either `'positive'` or `'negative'`.
- **source** (*set*) – Optional, `None` by default. If specified, determines which source(s) is(are) to be removed from the sources in the specified *direction* and *sign*.

which_directions(*resources=None*, *effect=None*)
Returns the pair(s) of nodes for which there is information about their directionality.

Parameters

- **effect** (*str*) – Either *positive* or *negative*.
- **resources** (*str,set*) – Limits the query to one or more resources. Optional.

Returns (*tuple*) – Tuple of tuples with pairs of nodes where the first element is the source and the second is the target entity, according to the given resources and limited to the effect.

which_dirs(*resources=None*, *effect=None*)
Returns the pair(s) of nodes for which there is information about their directionality.

Parameters

- **effect** (*str*) – Either *positive* or *negative*.
- **resources** (*str,set*) – Limits the query to one or more resources. Optional.

Returns (*tuple*) – Tuple of tuples with pairs of nodes where the first element is the source and the second is the target entity, according to the given resources and limited to the effect.

which_signs(*resources=None*, *effect=None*)
Returns the pair(s) of nodes for which there is information about their effect signs.

Parameters

- **resources** (*str,set*) – Limits the query to one or more resources. Optional.
- **effect** (*str*) – Either *positive* or *negative*, limiting the query to positive or negative effects; for any other values effects of both signs will be returned.

Returns (*tuple*) – Tuple of tuples with pairs of nodes where the first element is a tuple of the source and the target entity, while the second element is the effect sign, according to the given resources. E.g. ((('A', 'B'), 'positive'),)

class pypath.core.interaction.**InteractionDataFrameRecord**(*id_a*, *id_b*, *type_a*, *type_b*, *directed*, *effect*, *type*, *dmodel*, *sources*, *references*)

directed
Alias for field number 4

dmodel
Alias for field number 7

effect
Alias for field number 5

id_a
Alias for field number 0

id_b
Alias for field number 1

references
Alias for field number 9

sources
Alias for field number 8

type
Alias for field number 6

type_a
Alias for field number 2

type_b
Alias for field number 3

class pypath.core.interaction.**InteractionKey**(*entity_a*, *entity_b*)

entity_a
Alias for field number 0

entity_b
Alias for field number 1

2.19 intera

This module provides classes to represent and handle structural details of protein interactions i.e. residues, post-translational modifications, short motifs, domains, domain-motifs and domain-motif interactions, binding interfaces.

2.20 intercell_annot

pypath.core.intercell_annot.**go_single_terms = {'C': {'axolemma', 'banded collagen fibril',**
Higher level classes of intercellular communication roles.

2.21 intercell

class pypath.core.intercell.**IntercellRole**(*source*, *role*)

role
Alias for field number 1

source
Alias for field number 0

2.22 log

pypath.share.log.**new_logger**(*name=None*, *logdir=None*, *verbosity=None*, ***kwargs*)

Returns a new logger with default settings (can be customized).

name [str] Custom name for the log.

logdir [str] Path to the directoty to store log files.

verbosity [int] Verbosity level, lowest is 0. Messages from levels above this won't be written to the log..

`log.Logger` instance.

2.23 main

class pypath.legacy.main.**PyPath**(*ncbi_tax_id=None*, *copy=None*, *name='unnamed'*, *cache_dir=None*, *outdir='results'*, *loglevel=0*, *loops=False*)

This is the a *legacy* object representing a molecular interaction network. At some point it will be removed, we don't recommend to rely on it when you build your applications. The `pypath.network.Network` object offers a much clearer and more versatile API. As of end of 2019 not all functionalities have been migrated to the new API. For this reason we offer an intermediate solution: in this *igraph* based object the *attrs* edge attribute holds instances of `pypath.interaction.Interaction` objects, the same type of object we use to represent interactions in the new `pypath.network.Network`. At the same time we will keep supporting *igraph* with a method for converting `pypath.network.Network` to a `igraph.Graph` object, however this won't provide all the methods available here but will serve only the purpose to make it possible to use the graph theory methods from the *igraph* library on networks built with *pypath*.

An object representing a molecular interaction network.

Parameters

- **ncbi_tax_id** (*int*) – Optional, `9606` (Homo sapiens) by default. NCBI Taxonomic identifier of the organism from which the data will be downloaded.
- **default_name_type** (*dict*) – Optional, `{'protein': 'uniprot', 'mirna': 'mirbase', 'drug': 'chembl', 'lncrna': 'lncrna-genesymbol'}` by default. Contains the default identifier types to which the downloaded data will be converted. If others are used, user may need to provide the format definitions for the conversion tables.
- **copy** (*pypath.main.PyPath*) – Optional, `None` by default. Other `pypath.main.PyPath` instance from which the data will be copied.
- **name** (*str*) – Optional, `'unnamed'` by default. Session or project name (custom).
- **outdir** (*str*) – Optional, `'results'` by default. Output directory where to store all output files.
- **loglevel** (*int*) – Optional, 0 by default. Sets the level of the logger. The higher the level the more messages will be written to the log.
- **loops** (*bool*) – Optional, `False` by default. Determines if self-loop edges are allowed in the graph.

Variables

- **adjlist** (*list*) – List of [set] containing the adjacency of each node. See *PyPath.update_adjlist()* method for more information.

- **chembl** (*pypath.chembl.Chembl*) – Contains the ChEMBL data. See `pypath.chembl` module documentation for more information.
- **chembl_mysql** (*tuple*) – DEPRECATED Contains the MySQL parameters used by the `pypath.mapping` module to load the ChEMBL ID conversion tables.
- **data** (*dict*) – Stores the loaded interaction and attribute table. See `PyPath.read_data_file()` method for more information.
- **db_dict** (*dict*) – Dictionary of dictionaries. Outer-level keys are `'nodes'` and `'edges'`, corresponding values are [dict] whose keys are the database sources with values of type [set] containing the edge/node indexes for which that database provided some information.
- **dgraph** (*igraph.Graph*) – Directed network graph object.
- **disclaimer** (*str*) – Disclaimer text.
- **dlabDct** (*dict*) – Maps the directed graph node labels [str] (keys) to their indices [int] (values).
- **dnodDct** (*dict*) – Maps the directed graph node names [str] (keys) to their indices [int] (values).
- **dnodInd** (*set*) – Stores the directed graph node names [str].
- **dnodLab** (*dict*) – Maps the directed graph node indices [int] (keys) to their labels [str] (values).
- **dnodNam** (*dict*) – Maps the directed graph node indices [int] (keys) to their names [str] (values).
- **edgeAttrs** (*dict*) – Stores the edge attribute names [str] as keys and their corresponding types (e.g.: `set`, `list`, `str`, ...) as values.
- **exp** (*pandas.DataFrame*) – Stores the expression data for the nodes (if loaded).
- **exp_prod** (*pandas.DataFrame*) – Stores the edge expression data (as the product of the normalized expression between the pair of nodes by default). For more details see `pypath.main.PyPath.edges_expression()`.
- **exp_samples** (*set*) – Contains a list of tissues as downloaded by ProteomicsDB. See *PyPath.get_proteomicsdb()* for more information.
- **failed_edges** (*list*) – List of lists containing information about the failed edges. Each failed edge sublist contains (in this order): [tuple] with the node IDs, [str] names of nodes A and B, [int] IDs of nodes A and B and [int] IDs of the edges in both directions.
- ***go*** (*dict*) – Contains the organism(s) NCBI taxonomy ID as key [int] and `pypath.go.GOAnnotation` object as value, which contains the GO annotations for the nodes in the graph. See `pypath.go.GOAnnotation` for more information.
- **graph** (*igraph.Graph*) – Undirected network graph object.
- **gsea** (*pypath.gsea.GSEA*) – Contains the loaded gene-sets from MSigDB. See `pypath.gsea.GSEA` for more information.
- **has_cats** (*set*) – Contains the categories (e.g.: resource types) [str] loaded in the current network. Possible categories are: `'m'` for PTM/enzyme-substrate resources, `'p'` for pathway/activity flow resources, `'i'` for undirected/PPI resources, `'r'` for process description/reaction resources and `'t'` for transcription resources.
- **htp** (*dict*) – Contains information about high-throughput data of the network for different thresholds [int] (keys). Values are [dict] containing the number of references (`'rnum'`)

[int], number of edges (`'enum'`) [int], number of sources (`'snum'`) [int] and list of PMIDs of the most common references above the given threshold (`'htrefs'`) [set].

- **labDct** (*dict*) – Maps the undirected graph node labels [str] (keys) to their indices [int] (values).
- **lists** (*dict*) – Contains specific lists of nodes (values) for different categories [str] (keys). These can to be loaded from a file or a resource. Some methods include `pypath.main.PyPath.receptor_list()` (`'rec'`), `pypath.main.PyPath.druggability_list()` (`'dgb'`), `pypath.main.PyPath.kinases_list()` (`'kin'`), `pypath.main.PyPath.tfs_list()` (`'tf'`), `pypath.main.PyPath.disease_genes_list()` (`'dis'`), `pypath.main.PyPath.signaling_proteins_list()` (`'sig'`), `pypath.main.PyPath.proteome_list()` (`'proteome'`) and `pypath.main.PyPath.cancer_drivers_list()` (`'cdv'`).
- **loglevel** (*str*) – The level of the logger.
- **loops** (*bool*) – Whether if self-loop edges are allowed in the graph.
- **mapper** (*pypath.mapping.Mapper*) – `pypath.mapper.Mapper` object for ID conversion and other ID-related operations across resources.
- **mutation_samples** (*list*) – DEPRECATED
- **mysql_conf** (*tuple*) – DEPRECATED Contains the MySQL parameters used by the `pypath.mapping` module to load the ID conversion tables.
- ***name*** (*str*) – Session or project name (custom).
- **ncbi_tax_id** (*int*) – NCBI Taxonomic identifier of the organism from which the data will be downloaded.
- **negatives** (*dict*) – Contains a list of negative interactions according to a given source (e.g.: Negatome database). See `pypath.main.PyPath.apply_negative()` for more information.
- **nodDct** (*dict*) – Maps the undirected graph node names [str] (keys) to their indices [int] (values).
- **nodInd** (*set*) – Stores the undirected graph node names [str].
- **nodLab** (*dict*) – Maps the undirected graph node indices [int] (keys) to their labels [str] (values).
- **nodNam** (*dict*) – Maps the directed graph node indices [int] (keys) to their names [str] (values).
- **outdir** (*str*) – Output directory where to store all output files.
- **palette** (*list*) – Contains a list of hexadecimal [str] of colors. Used for plotting purposes.
- **pathway_types** (*list*) – Contains the names of all the loaded pathway resources [str].
- **pathways** (*dict*) – Contains the list of pathways (values) for each resource (keys) loaded in the network.
- **plots** (*dict*) – DEPRECATED (?)
- **proteomicsdb** (*pypath.proteomicsdb.ProteomicsDB*) – Contains a `pypath.proteomicsdb.ProteomicsDB` instance, see the class documentation for more information.

- **raw_data** (*list*) – Contains a list of loaded edges [dict] from a data file. See `PyPath.read_data_file()` for more information.
- ***seq*** (*dict*) – (?)
- ***session*** (*str*) – Session ID, a five random alphanumeric characters.
- **session_name** (*str*) – Session name and ID (e.g. `'unnamed-abc12'`).
- **sourceNetEdges** (*igraph.Graph*) – (?)
- **sourceNetNodes** (*igraph.Graph*) – (?)
- ***sources*** (*list*) – List contianing the names of the loaded resources [str].
- **u_pfam** (*dict*) – Dictionary of dictionaries, contains the mapping of UniProt IDs to their respective protein families and other information.
- **uniprot_mapped** (*list*) – DEPRECATED (?)
- **unmapped** (*list*) – Contains the names of unmapped items [str]. See `pypath.main.PyPath.map_item()` for more information.
- **vertexAttrs** (*dict*) – Stores the node attribute names [str] as keys and their corresponding types (e.g.: `set`, `list`, `str`, ...) as values.

acsn_effects(*graph=None*)

add_genesets(*genesets*)

add_grouped_eattr(*edge*, *attr*, *group*, *value*)
Merges (or creates) a given edge attribute as [dict] of [list] values.

Parameters

- **edge** (*int*) – Edge index where the given attribute value is to be merged or created.
- **attr** (*str*) – The name of the attribute. If such attribute does not exist in the network edges, it will be created on all edges (as an empty [dict], *value* will only be assigned to the given *edge* and *group*).
- **group** (*str*) – The key of the attribute dictionary where *value* is to be assigned.
- **value** (*list*) – The value of the attribute to be assigned/merged.

add_grouped_set_eattr(*edge*, *attr*, *group*, *value*)
Merges (or creates) a given edge attribute as [dict] of [set] values.

Parameters

- **edge** (*int*) – Edge index where the given attribute value is to be merged or created.
- **attr** (*str*) – The name of the attribute. If such attribute does not exist in the network edges, it will be created on all edges (as an empty [dict], *value* will only be assigned to the given *edge* and *group*).
- **group** (*str*) – The key of the attribute dictionary where *value* is to be assigned.
- **value** (*set*) – The value of the attribute to be assigned/merged.

add_list_eattr(*edge*, *attr*, *value*)
Merges (or creates) a given edge attribute as [list].

Parameters

- **edge** (*int*) – Edge index where the given attribute value is to be merged or created.

- **attr** (*str*) – The name of the attribute. If such attribute does not exist in the network edges, it will be created on all edges (as an empty [list], *value* will only be assigned to the given *edge*).
- **value** (*list*) – The value of the attribute to be assigned/merged.

add_set_eattr(*edge*, *attr*, *value*)
Merges (or creates) a given edge attribute as [set].

Parameters

- **edge** (*int*) – Edge index where the given attribute value is to be merged or created.
- **attr** (*str*) – The name of the attribute. If such attribute does not exist in the network edges, it will be created on all edges (as an empty [set], *value* will only be assigned to the given *edge*).
- **value** (*set*) – The value of the attribute to be assigned/merged.

affects(*identifier*)

all_between(*id_a*, *id_b*)
Checks for any edges (in any direction) between the provided nodes.

Parameters

- **id_a** (*str*) – The name of the source node.
- **id_b** (*str*) – The name of the target node.

Returns (*dict*) – Contains information on the directionality of the requested edge. Keys are `'ab'` and `'ba'`, denoting the straight/reverse directionalities respectively. Values are [list] whose elements are the edge ID or `None` according to the existance of that edge in the following categories: undirected, straight and reverse (in that order).

all_neighbours(*indices=False*)
Looks for the first neighbours of all the nodes and creates an attribute (`'neighbours'`) on each one of them containing a list of their UniProt IDs.

Parameters **indices** (*bool*) – Optional, `False` by default. Whether to list the neighbour nodes indices or their UniProt IDs.

apply_list(*name*, *node_or_edge='node'*)
Creates vertex or edge attribute based on a list.

Parameters

- **name** (*str*) – The name of the list to be added as attribute. Must have been previously loaded with `pypath.main.PyPath.load_list()` or other methods. See description of `pypath.main.PyPath.lists` attribute for more information.
- **node_or_edge** (*str*) – Optional, `'node'` by default. Whether the attribute list is to be added to the nodes or to the edges.

apply_negative(*settings*)
Loads a negative interaction source (e.g.: Negatome) into the current network.

Parameters **settings** (*pypath.input_formats.NetworkInput*) – `pypath.input_formats.NetworkInput` instance containing the detailed definition of the input format to the downloaded file. For instance `pypath.data_formats.negative['negatome']`

basic_stats(*latex=False, caption='', latex_hdr=True, fontsize=8, font='HelveticaNeueLTStd-LtCn', fname=None, header_format='%s', row_order=None, by_category=True, use_cats=['p', 'm', 'i', 'r'], urls=True, annots=False*)
Returns basic numbers about the network resources, e.g. edge and node counts.

latex Return table in a LaTeX document. This can be compiled by PDFLaTeX: latex stats.tex

basic_stats_intergroup(*groupA*, *groupB*, *header=None*)

cancer_drivers_list(*intogen_file=None*)
Loads the list of cancer drivers. Contains information from COSMIC (needs user log in credentials) and IntOGen (if provided) and adds the attribute to the undirected network nodes.

Parameters **intogen_file** (*str*) – Optional, `None` by default. Path to the data file. Can also be [function] that provides the data. In general, anything accepted by `pypath.input_formats.NetworkInput.input`.

cancer_gene_census_list()
Loads the list of cancer driver proteins from the COSMIC Cancer Gene Census.

clean_graph(*organisms_allowed=None*)
Removes multiple edges, unknown molecules and those from wrong taxon. Multiple edges will be combined by `pypath.main.PyPath.combine_attr()` method. Loops will be deleted unless the attribute `pypath.main.PyPath.loops` is set to `True`.

Parameters **organisms_allowed** (*set*) – NCBI Taxonomy identifiers [int] of the organisms allowed in the network.

collapse_by_name(*graph=None*)
Collapses nodes with the same name by copying and merging all edges and attributes. Operates directly on the provided network object.

Parameters **graph** (*igraph.Graph*) – Optional, `None` by default. The network for which the nodes are to be collapsed. If none is provided, takes `pypath.main.PyPath.graph` (undirected network) by default.

collect(*method*, ***kwargs*)
Collects various entities over the network according to `method`.

combine_attr(*lst*, *num_method=<built-in function max>*)
Combines multiple attributes into one. This method attempts to find out which is the best way to combine attributes.

- If there is only one value or one of them is None, then returns the one available.
- Lists: concatenates unique values of lists.
- Numbers: returns the greater by default or calls *num_method* if given.
- Sets: returns the union.
- Dictionaries: calls `pypath.common.merge_dicts()`.
- Direction: calls their special `pypath.main.Direction.merge()` method.

Works on more than 2 attributes recursively.

Parameters

- **lst** (*list*) – List of one or two attribute values.
- **num_method** (*function*) – Optional, `max` by default. Method to merge numeric attributes.

communities(*method*, ***kwargs*)

complex_comembership_network(*graph=None*, *resources=None*)

complexes(*methods=['3dcomplexes', 'havugimana', 'corum', 'complexportal', 'compleat']*)

complexes_in_network(*csource='corum'*, *graph=None*)

compounds_from_chembl(*chembl_mysql=None, nodes=None, crit=None, andor='or', assay_types=['B', 'F'], relationship_types=['D', 'H'], multi_query=False, **kwargs*)

Loads compound data from ChEMBL to the network.

Parameters

- **chebl_mysql** (*tuple*) – Optional, `None` by default. Contains the MySQL parameters used by the `pypath.mapping` module to loadthe ChEMBL ID conversion tables. If none is passed, takes the current instance `pypath.main.PyPath.chembl_mysql` attribute.
- **nodes** (*list*) – Optional, `None` by default. List of node indices for which the information is to be loaded. If none is provided calls the method `pypath.main.PyPath.get_sub()` with the provided *crit* parameter.
- **crit** (*dict*) – Optional, `None` by default. Defines the critical attributes to generate a subnetwork to extract the nodes in case *nodes* is not provided. Keys are `'edge'` and `'node'` and values are [dict] containing the critical attribute names [str] and values are [set] containing those attributes of the nodes/edges that are to be kept in the subnetwork. If none is provided, takes the whole network.
- **andor** (*str*) – Optional, `'or'` by default. Determines the search mode for the subnetwork generation (if `nodes=None`). See `pypath.main.PyPath.search_attr_or()` and `pypath.main.PyPath.search_attr_and()` for more details.
- **assay_types** (*list*) – Optional, `['B', 'F']` by default. Types of assay to query Options are: `'A'` (ADME), `'B'` (Binding),‘‘F’‘ (Functional), `'P'` (Physicochemical), `'T'` (Toxicity) and/or `'U'` (Unassigned).
- **relationship_types** (*list*) – Optional, `['D', 'H']` by default. Assay relationship types to query. Possible values are: `'D'` (Direct protein target assigned), `'H'` (Homologous protein target assigned), `'M'` (Molecular target other than protein assigned), `'N'` (Non-molecular target assigned), `'S'` (Subcellular target assigned) and/or `'U'` (Default value, target has yet to be curated).
- **multi_query** (*bool*) – Optional, `False` by default. Not used.
- ****kwargs** – Additional keyword arguments for `pypath.chembl.Chembl.compounds_targets()`.

consistency()

copy(*other*)

Copies another `pypath.main.PyPath` instance into the current one.

Parameters **other** (*pypath.main.PyPath*) – The instance to be copied from.

copy_edges(*sources*, *target*, *move=False*, *graph=None*)

Copies edges from *sources* node(s) to another one (*target*), keeping attributes and directions.

Parameters

- **sources** (*list*) – Contains the vertex index(es) [int] of the node(s) to be copied or moved.
- **target** (*int*) – Vertex index where edges and attributes are to be copied to.

- **move** (*bool*) – Optional, `False` by default. Whether to perform copy or move (remove or keep the source edges).
- **graph** (*igraph.Graph*) – Optional, `None` by default. The network graph object from which the nodes are to be merged. If none is passed, takes the undirected network graph.

count_sol()
Counts the number of nodes with zero degree.

Returns (*int*) – The number of nodes with zero degree.

counts(*collection_method*, *add_total=True*, *add_percent=True*, *add_cat_total=True*, ***kwargs*)
Collects various entities over the network according to `method` and counts them in total and by resources.

coverage(*lst*)
Computes the coverage (range [0, 1]) of a list of nodes against the current (undirected) network.

Parameters **lst** (*set*) – Can also be [list] (will be converted to [set]) or [str]. In the latter case it will retrieve the list with that name (if such list exists in `pypath.main.PyPath.lists`).

curation_effort(*resources=None*, ***kwargs*)
Returns a *set* of reference-interactions pairs.

curation_effort_by_resource(*resources=None*, ***kwargs*)
A *dict* with resources as keys and *set*s of curation items (interaction-reference pairs) as values.

curation_stats(*by_category=True*)

curation_tab(*fname='curation_stats.tex'*, *by_category=True*, *use_cats=['p'*, *'m'*, *'i'*, *'r']*, *header_size='normalsize'*, ***kwargs*)

curators_work()
Computes and prints an estimation of how many years of curation took to achieve the amount of information on the network.

databases_similarity(*index='simpson'*)
Computes the similarity across databases according to a given index metric. Computes the similarity across the loaded resources (listed in `pypath.main.PyPath.sources` in terms of nodes and edges separately.

Parameters **index** (*str*) – Optional, `'simpson'` by default. The type of index metric to use to compute the similarity. Options are `'simpson'`, `'sorensen'` and `'jaccard'`.

Returns (*dict*) – Nested dictionaries (three levels). First-level keys are `'nodes'` and `'edges'`, then second and third levels correspond to sources names which map to the similarity index between those sources [float].

degree_dist(*prefix*, *g=None*, *group=None*)
Computes the degree distribution over all nodes of the network. If *group* is provided, also across nodes of that group(s).

Parameters

- **prefix** (*str*) – Prefix for the file name(s).
- **g** (*igraph.Graph*) – Optional, `None` by default. The network over which to compute the degree distribution. If none is passed, takes the undirected network of the current instance.
- **group** (*list*) – Optional, `None` by default. Additional group(s) name(s) [str] of node attributes to subset the network and compute its degree distribution.

degree_dists()
Computes the degree distribution for all the different network sources. This is, for each source, the subnetwork comprising all interactions coming from it is extracted and the degree distribution information is computed and saved into a file. A file is created for each resource under the name `'pwnet-<session_id>-degdist-<resource>''`. Files are stored in `pypath.main.PyPath.outdir` (`'results'` by default).

delete_by_organism(*organisms_allowed=None*)
Removes the proteins of all organisms which are not given in *tax*.

Parameters **organisms_allowed** (*list, set*) – List of NCBI Taxonomy IDs [int] of the organism(s) that are to be kept.

delete_by_source(*source*, *vertexAttrsToDel=None*, *edgeAttrsToDel=None*)
Deletes nodes and edges from the network according to a provided source name. Optionally can also remove the given list of attributes from nodes and/or edges.

Parameters

- **source** (*str*) – Name of the source from which the nodes and edges have to be removed.
- **vertexAttrsToDel** (*list*) – Optional, `None` by default. Contains the names [str] of the attributes to be removed from the nodes.
- **edgeAttrsToDel** (*list*) – Optional, `None` by default. Contains the names [str] of the attributes to be removed from the edges.

delete_unknown(*organisms_allowed=None*, *entity_type='protein'*, *default_name_type=None*)
Removes those items which are not in the list of all default IDs of the organisms. By default, it means to remove all protein nodes not having a human UniProt ID.

Parameters

- **typ** (*str*) – Optional, `'protein'` by default. Determines the molecule type. These can be `'protein'`, `'drug'`, `'lncrna'`, `'mirna'` or any other type defined in `pypath.main.PyPath.default_name_type`.
- **default_name_type** (*str*) – Optional, `None` by default. The default name type for the given molecular species. If none is specified takes it from `pypath.main.PyPath.default_name_type` (e.g.: for `'protein'`, default is `'uniprot'`).
- **organisms_allowed** (*set*) – NCBI Taxonomy identifiers [int] of the organisms allowed in the network.

delete_unmapped()
Checks the network for any existing unmapped node and removes it.

dgenesymbol(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default directed network, otherwise `None`.

@genesymbol [str] GeneSymbol.

dgenesymbols(*genesymbols*)

dgs(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default directed network, otherwise `None`.

@genesymbol [str] GeneSymbol.

dgss(*genesymbols*)

dneighbors(*identifier*, *mode='ALL'*)

dp(*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

dproteins(*identifiers*)

dps(*identifiers*)

duniprot(*uniprot*)
Same as `PyPath.uniprot(), just for directed graph. Returns ``igraph.Vertex()` object if the UniProt can be found in the default directed network, otherwise `None`.

@uniprot [str] UniProt ID.

duniprots(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default directed graph.

dup(*uniprot*)
Same as `PyPath.uniprot(), just for directed graph. Returns ``igraph.Vertex()` object if the UniProt can be found in the default directed network, otherwise `None`.

@uniprot [str] UniProt ID.

dups(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default directed graph.

dv(*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

dvs(*identifiers*)

edge_exists(*id_a*, *id_b*)
Returns a tuple of vertex indices if edge doesn't exist, otherwise, the edge ID. Not sensitive to direction.

Parameters

- **id_a** (*str*) – Name of the source node.
- **id_b** (*str*) – Name of the target node.

Returns (*int*) – The edge index, if exists such edge. Otherwise, [tuple] of [int] corresponding to the node IDs.

edge_loc(*graph=None*, *topn=2*)

edge_names(*e*)
Returns the node names of a given edge.

Parameters **e** (*int*) – The edge index.

Returns (*tuple*) – Contains the source and target node names of the edge [str].

edges_3d(*methods=['dataio.get_instruct', 'dataio.get_i3d']*)

edges_between(*group1*, *group2*, *directed=True*, *strict=False*)
Selects edges between two groups of vertex IDs. Returns set of edge IDs.

Parameters

- **group1,group2** (*set*) – List, set or tuple of vertex IDs.
- **directed** (*bool*) – Only edges with direction *group1 -> group2* selected.
- **strict** (*bool*) – Edges with no direction information still selected even if `directed` is *False*.

edges_expression(*func=<function PyPath.<lambda>>*)
Executes function *func* for each pairs of connected proteins in the network, for every expression dataset. By default, *func* simply gives the product the (normalized) expression values.

func [callable] Function to handle 2 vectors (pandas.Series() objects), should return one vector of the same length.

edges_in_complexes(*csources=['corum'], graph=None*)
Creates edge attributes `complexes` and `in_complex`. These are both dicts where the keys are complex resources. The values in `complexes` are the list of complex names both the source and the target vertices belong to. The values `in_complex` are boolean values whether there is at least one complex in the given resources both the source and the target vertex of the edge belong to.

@csources [list] List of complex resources. Should be already loaded.

@graph [igraph.Graph()] The graph object to do the calculations on.

edges_ptms()

edgeseq_inverse(*edges*)
Returns the sequence of all edge indexes that are not in the argument *edges*.

Parameters **edges** (*set*) – Sequence of edge indices [int] that will not be returned.

Returns (*list*) – Contains all edge indices [int] of the undirected network except the ones on *edges* argument.

entities_by_resource(*resources=None*, *entity_type=None*, ***kwargs*)
Returns a *dict* of *set*s with resources as keys and sets of entities as values.

entities_by_resources()
Returns a dict of sets with resources as keys and sets of entity IDs as values.

export_dot(*nodes=None*, *edges=None*, *directed=True*, *labels='genesymbol'*, *edges_filter=<function PyPath.<lambda>>*, *nodes_filter=<function PyPath.<lambda>>*, *edge_sources=None*, *dir_sources=None*, *graph=None*, *return_object=False*, *save_dot=None*, *save_graphics=None*, *prog='neato'*, *format=None*, *hide=False*, *font=None*, *auto_edges=False*, *hide_nodes=[]*, *defaults={}*, ***kwargs*)
Builds a pygraphviz.AGraph() object with filtering the edges and vertices along arbitrary criteria. Returns the Agraph object if requesred, or exports the dot file, or saves the graphics.

@nodes : list List of vertex ids to be included. @edges : list List of edge ids to be included. @directed : bool Create a directed or undirected graph. @labels : str Name type to be used as id/label in the dot format. @edges_filter : function Function to filter edges, accepting igraph.Edge as argument. @nodes_filter : function Function to filter vertices, accepting igraph.Vertex as argument. @edge_sources : list Sources to be included. @dir_sources : list Direction and effect sources to be included. @graph : igraph.Graph The graph object to export. @return_object : bool Whether to return the pygraphviz.AGraph object. @save_dot : str Filename to export the dot file to. @save_graphics : str Filename to export the graphics, the extension defines the format. @prog : str The graphviz layout algorithm to use. @format : str The graphics format passed to pygraphviz.AGrapg().draw(). @hide : bool Hide filtered edges instead of omit them. @hide nodes : list Nodes to hide. List of vertex ids. @auto_edges : str Automatic, built-in

style for edges. 'DIRECTIONS' or 'RESOURCE_CATEGORIES' are supported. @font : str Font to use for labels. For using more than one fonts refer to graphviz attributes with constant values or define callbacks or mapping dictionaries. @defaults : dict Default values for graphviz attributes, labeled with the entity, e.g. *{'edge_penwidth': 0.2}*. @**kwargs : constant, callable or dict Graphviz attributes, labeled by the target entity. E.g. *edge_penwidth*, 'vertex_shape' or *graph_label*. If the value is constant, this value will be used. If the value is dict, and has *_name* as key, for every instance of the given entity, the value of the attribute defined by *_name* will be looked up in the dict, and the corresponding value will be given to the graphviz attribute. If the key *_name* is missing from the dict, igraph vertex and edge indices will be looked up among the keys. If the value is callable, it will be called with the current instance of the entity and the returned value will be used for the graphviz attribute. E.g. *edge_arrowhead(edge)* or *vertex_fillcolor(vertex)* Example:

import pypath from pypath import data_formats net = pypath.PyPath() net.init_network(pfile = 'cache/default.pickle') #net.init_network({'arn': data_formats.omnipath['arn']}) tgf = [v.index for v in net.graph.vs if 'TGF' in v['slk_pathways']] dot = net.export_dot(nodes = tgf, save_graphics = 'tgf_slk.pdf', prog = 'dot',

main_title = 'TGF-beta pathway', return_object = True, label_font = 'HelveticaNeueLT-Std Med Cn', edge_sources = ['SignaLink3'], dir_sources = ['SignaLink3'], hide = True)

export_edgelist (*fname, graph=None, names=['name'], edge_attributes=[], sep='\t'*)
Write edge list to text file with attributes

Parameters

- **fname** – the name of the file or a stream to read from.
- **graph** – the igraph object containing the network
- **names** – list with the vertex attribute names to be printed for source and target vertices
- **edge_attributes** – list with the edge attribute names to be printed
- **sep** – string used to separate columns

export_graphml (*outfile=None*, *graph=None*, *name='main'*)
Saves the network in a `.graphml` file.

Parameters

- **outfile** (*str*) – Optional, `None` by default. Name/path of the output file. If none is passed, `'results/netrowk-<session_id>.graphml'` is used.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The network object to be saved. If none is passed, takes the undirected network of the current instance.
- **name** (*str*) – Optional, `'main'` by default. The graph name for the output file.

export_ptms_tab (*outfile=None*)
Exports a tab file containing the PTM interaction information loaded in the network.

Parameters **outfile** (*str*) – Optional, `None` by default. The output file nama/path to store the PTM information. If none is provided, the default is `'results/network-<session_id>.tab'`.

Returns (*list*) – Contains the edge indices [int] of all PTM interactions.

export_sif (*outfile=None*)
Exports the network interactions in `.sif` format (Simple Interaction Format).

Parameters **outfile** (*str*) – Optional, `None` by default. Name/path of the output file. If none is passed, `'results/netrowk-<session_id>.sif'` is used.

export_struct_tab(*outfile=None*)
Exports a tab file containing the domain interaction information and PTM regulation loaded in the network.

Parameters **outfile** (*str*) – Optional, `None` by default. The output file nama/path to store the PTM information. If none is provided, the default is `'results/network-<session_id>.tab'`.

Returns (*list*) – Contains the edge indices [int] of all PTM interactions.

export_tab(*outfile=None*, *extra_node_attrs={}*, *extra_edge_attrs={}*, *unique_pairs=True*, ***kwargs*)
Exports the network in a tabular format. By default UniProt IDs, Gene Symbols, source databases, literature references, directionality and sign information and interaction type are included.

Parameters

- **outfile** (*str*) – Optional, `None` by default. Name/path of the output file. If none is passed, `'results/netrowk-<session_id>.tab'` is used.
- **extra_node_attrs** (*dict*) – Optional, `{}` by default. Additional node attributes to be included in the exported table. Keys are column names used in the header while values are names of vertex attributes. In the header `'_A'` and `'_B'` suffixes will be appended to the column names so the values can be assigned to A and B side interaction partners.
- **extra_edge_attrs** (*dict*) – Optional, `{}` by default. Additional edge attributes to be included in the exported table. Keys are column names used in the header while values are names of edge attributes.
- **unique_pairs** (*bool*) – Optional, `True` by default. If set to `True` each line corresponds to a unique pair of molecules, all directionality and sign information are covered in other columns. If `False`, order of `'A'` and `'B'` IDs corresponds to the direction while sign covered in further columns.
- **kwargs** (****) – Additional keyword arguments passed to `pypath.export.Export`.

find_all_paths(*start*, *end*, *attr=None*, *mode='OUT'*, *maxlen=2*, *graph=None*, *silent=False*, *update_adjlist=True*)
Finds all paths up to length *maxlen* between groups of vertices. This function is needed only becaues igraph's get_all_shortest_paths() finds only the shortest, not any path up to a defined length.

start [int or list] Indices of the starting node(s) of the paths.

end [int or list] Indices of the target node(s) of the paths.

attr [str] Name of the vertex attribute to identify the vertices by. Necessary if `start` and `end` are not igraph vertex ids but for example vertex names or labels.

mode ['IN', 'OUT', 'ALL'] Passed to igraph.Graph.neighbors()

maxlen [int] Maximum length of paths in steps, i.e. if maxlen = 3, then the longest path may consist of 3 edges and 4 nodes.

graph [igraph.Graph object] The graph you want to find paths in. self.graph by default.

find_all_paths2(*graph*, *start*, *end*, *mode='OUT'*, *maxlen=2*, *psize=100*, *update_adjlist=True*)

find_complex(*search*)
Finds complexes by their non standard names. E.g. to find DNA polymerases you can use the search term *DNA pol* which will be tested against complex names in CORUM.

first_neighbours(*node*, *indices=False*)
Looks for the first neighbours of a given node and returns a list of their UniProt IDs.

Parameters

- **node** (*str*) – The UniProt ID of the node of interest. Can also be the index of such node [int].
- **indices** (*bool*) – Optional, `False` by default. Whether to return the neighbour nodes indices or their UniProt IDs.

Returns (*list*) – The list containing the first neighbours of the queried node.

fisher_enrichment (*lst*, *attr*, *ref='proteome'*)

Computes an enrichment analysis using Fisher's exact test. The contingency table is built as follows: First row contains the number of nodes in the *ref* list (such list is considered to be loaded in `pypath.main.PyPath.lists`) and the number of nodes in the current (undirected) network. Second row contains the number of nodes in *lst* list (also considered to be already loaded) and the number of nodes in the network with a non-empty attribute *attr*. Uses `scipy.stats.fisher_exact()`, see the documentation of the corresponding package for more information.

Parameters

- **lst** (*str*) – Name of the list in `pypath.main.PyPath.lists` whose number of elements will be the first element in the second row of the contingency table.
- **attr** (*str*) – The node attribute name for which the number of nodes in the network with such attribute will be the second element of the second row of the contingency table.
- **ref** (*str*) – Optional, `'proteome'` by default. The name of the list in `pypath.main.PyPath.lists` whose number of elements will be the first element of the first row of the contingency table.

Returns

- (*float*) – Prior odds ratio.
- (*float*) – P-value or probability of obtaining a distribution as extreme as the observed, assuming that the null hypothesis is true.

geneset_enrichment (*proteins*, *all_proteins=None*, *geneset_ids=None*, *alpha=0.05*, *correction_method='hommel'*)

Does not work at the moment because cfisher module should be replaced with scipy.

genesymbol (*genesymbol*)

Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default undirected network, otherwise `None`.

@genesymbol [str] GeneSymbol.

genesymbol_labels (*graph=None*, *remap_all=False*)

Creats vertex attribute `'label'` and fills up with the corresponding GeneSymbols of all proteins where the GeneSymbol can be looked up based on the default name of the protein vertex (UniProt ID by default). If the attribute `'label'` has been already initialized, updates this attribute or recreates if *remap_all* is set to `True`.

Parameters

- **graph** (*igraph.Graph*) – Optional, `None` by default. The network graph object where the GeneSymbol labels are to be set/updated. If none is passed, takes the current network undirected graph by default (`pypath.main.PyPath.graph`).
- **remap_all** (*bool*) – Optional, `False` by default. Whether to map anew the GeneSymbol labels if those were already initialized.

genesymbols (*genesymbols*)

get_attrs (*line*, *spec*, *lnum*)

get_directed(*graph=False*, *conv_edges=False*, *mutual=False*, *ret=False*)
Converts a copy of *graph* undirected *igraph.Graph* object to a directed one. By default it converts the current network instance in `pypath.main.PyPath.graph` and places the copy of the directed instance in `pypath.main.PyPath.dgraph`.

Parameters

- **graph** (*igraph.Graph*) – Optional, `None` by default. Undirected graph object. If none is passed, takes the current undirected network instance and saves the directed network under the attribute `pypath.main.PyPath.dgraph`. Otherwise, the directed graph will be returned instead.
- **conv_edges** (*bool*) – Optional, `False` by default. Whether to convert undirected edges (those without explicit direction information) to an arbitrary direction edge or a pair of opposite edges. Otherwise those will be deleted.
- **mutual** (*bool*) – Optional, `False` by default. If *conv_edges* is `True`, whether to convert the undirected edges to a single, arbitrary directed edge, or a pair of opposite directed edges.
- **ret** (*bool*) – Optional, `False` by default. Whether to return the directed graph instance, or not. If a *graph* is provided, its directed version will be returned anyway.

Returns (*igraph.Graph*) – If *graph* is passed or *ret* is `True`, returns the copy of the directed graph. otherwise returns `None`.

get_dirs_signs()

get_edge(*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

Parameters

- **source** (*int, str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **target** (*int, str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **directed** (*bool*) – To be passed to igraph.Graph.get_eid()

get_edges(*sources*, *targets*, *directed=True*)
Returns a generator with all edges between source and target vertices.

Parameters

- **sources** (*iterable*) – Source vertex IDs, names, labels, or any iterable yielding `igraph.Vertex` objects.
- **targets** (*iterable*) – Target vertec IDs, names, labels, or any iterable yielding `igraph.Vertex` objects.
- **directed** (*bool*) – Passed to `igraph.get_eid()`.

get_function(*fun*)

get_giant(*replace=False*, *graph=None*)
Returns the giant component of the *graph*, or replaces the `igraph.Graph` instance with only the giant component if specified.

Parameters

- **replace** (*bool*) – Optional, `False` by default. Specifies whether to replace the `igraph.Graph` instance. This can be either the undirected network of the current

pypath.main.PyPath instance (default) or the one passed under the keyword argument *graph*.

- **graph** (*igraph.Graph*) – Optional, None by default. The graph object from which the giant component is to be computed. If none is specified, takes the undirected network of the current pypath.main.PyPath instance.

Returns (*igraph.Graph*) – If replace=False, returns a copy of the giant component graph.

get_go(*organism=None*)
Returns the GOAnnotation object for the organism requested (or the default one).

get_max(*attrList*)

get_network(*crit*, *andor='or'*, *graph=None*)
Retrieves a subnetwork according to a set of user-defined attributes. Basically applies pypath.main.PyPath.get_sub() on a given *graph*.

Parameters

- **crit** (*dict*) – Defines the critical attributes to generate the subnetwork. Keys are 'edge' and 'node' and values are [dict] containing the critical attribute names [str] and values are [set] containing those attributes of the nodes/edges that are to be kept.
- **andor** (*str*) – Optional, 'or' by default. Determines the search mode. See pypath.main.PyPath.search_attr_or() and pypath.main.PyPath.search_attr_and() for more details.
- **graph** (*igraph.Graph*) – Optional, None by default. The graph object where to extract the subnetwork. If none is passed, takes the current network (undirected) graph (pypath.main.PyPath.graph).

Returns (*igraph.Graph*) – The subgraph obtained from filtering according to the attributes defined in *crit*.

get_node(*identifier*)
Returns igraph.Vertex() object if the identifier is a valid vertex index in the default undirected graph, or a UniProt ID or GeneSymbol which can be found in the default undirected network, otherwise None.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or igraph.Vertex object.

get_node_d(*identifier*)
Same as PyPath.get_node, just for the directed graph. Returns igraph.Vertex() object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise None.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or igraph.Vertex object.

get_node_pair(*id_a*, *id_b*, *directed=False*)
Retrieves the node IDs from a pair of node names.

Parameters

- **id_a** (*str*) – Name of the source node.
- **id_b** (*str*) – Name of the target node.
- **directed** (*bool*) – Optional, False by default. Whether to return the node indices from the directed or undirected graph.

Returns (*tuple*) – The pair of node IDs of the selected graph. If not found, returns False.

get_nodes(*identifiers*)

get_nodes_d(*identifiers*)

get_pathways(*source*)

get_proteomicsdb(*user*, *passwd*, *tissues=None*, *pickle=None*)

get_sub(*crit*, *andor='or'*, *graph=None*)
Selects the nodes from *graph* (and edges to be removed) according to a set of user-defined attributes.

Parameters

- **crit** (*dict*) – Defines the critical attributes to generate the subnetwork. Keys are `'edge'` and `'node'` and values are [dict] containing the critical attribute names [str] and values are [set] containing those attributes of the nodes/edges that are to be kept.
- **andor** (*str*) – Optional, `'or'` by default. Determines the search mode. See `pypath.main.PyPath.search_attr_or()` and `pypath.main.PyPath.search_attr_and()` for more details.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object where to extract the subnetwork. If none is passed, takes the current network (undirected) graph (`pypath.main.PyPath.graph`).

Returns (*dict*) – Keys are `'nodes'` and `'edges'` whose values are [lst] of elements (as indexes [int]). Nodes are those to be kept and edges to be removed on the extracted subnetwork.

get_taxon(*tax_dict*, *fields*)

go_annotate_graph(*aspects=('C', 'F', 'P')*)
Annotates protein nodes with GO terms. In the `go` vertex attribute each node is annotated by a dict of sets where keys are one letter codes of GO aspects and values are sets of GO accessions.

go_enrichment(*proteins=None*, *aspect='P'*, *alpha=0.05*, *correction_method='hommel'*, *all_proteins=None*)
Does not work at the moment because cfisher module should be replaced with scipy.

gs(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default undirected network, otherwise `None`.

@genesymbol [str] GeneSymbol.

gs_affected_by(*genesymbol*)

gs_affects(*genesymbol*)

gs_edge(*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

@source [str] GeneSymbol

@target [str] GeneSymbol

@directed [bool] To be passed to igraph.Graph.get_eid()

gs_in_directed(*genesymbol*)

gs_in_undirected(*genesymbol*)

gs_inhibited_by(*genesymbol*)

gs_inhibits(*genesymbol*)

gs_neighborhood(*genesymbols*, *order=1*, *mode='ALL'*)

gs_neighbors(*genesymbol*, *mode='ALL'*)

gs_stimulated_by(*genesymbol*)

gs_stimulates(*genesymbol*)

gss(*genesymbols*)

guide2pharma()

having_attr(*attr*, *graph=None*, *index=True*, *edges=True*)
Checks if edges or nodes of the network have a specific attribute and returns an iterator of the indices (or the edge/node instances) of edges/nodes having such attribute.

Parameters

- **attr** (*str*) – The name of the attribute to look for.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object where the edge/node attribute is to be searched. If none is passed, takes the undirected network of the current instance.
- **index** (*bool*) – Optional, `True` by default. Whether to return the iterator of the indices or the node/edge instances.
- **edges** (*bool*) – Optional, `True` by default. Whether to look for the attribute in the networks edges or nodes instead.

Returns (*generator*) – Generator object containing the edge/node indices (or instances) having the specified attribute.

having_eattr(*attr*, *graph=None*, *index=True*)
Checks if edges of the network have a specific attribute and returns an iterator of the indices (or the edge instances) of edges having such attribute.

Parameters

- **attr** (*str*) – The name of the attribute to look for.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object where the edge/node attribute is to be searched. If none is passed, takes the undirected network of the current instance.
- **index** (*bool*) – Optional, `True` by default. Whether to return the iterator of the indices or the node/edge instances.

Returns (*generator*) – Generator object containing the edge indices (or instances) having the specified attribute.

having_ptm(*index=True*, *graph=None*)
Checks if edges of the network have the `'ptm'` attribute and returns an iterator of the indices (or the edge instances) of edges having such attribute.

Parameters

- **index** (*bool*) – Optional, `True` by default. Whether to return the iterator of the indices or the node/edge instances.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object where the edge/node attribute is to be searched. If none is passed, takes the undirected network of the current instance.

Returns (*generator*) – Generator object containing the edge indices (or instances) having the `ptm''` attribute.

having_vattr(*attr*, *graph=None*, *index=True*)
Checks if nodes of the network have a specific attribute and returns an iterator of the indices (or the node instances) of nodes having such attribute.

Parameters

- **attr** (*str*) – The name of the attribute to look for.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object where the edge/node attribute is to be searched. If none is passed, takes the undirected network of the current instance.
- **index** (*bool*) – Optional, `True` by default. Whether to return the iterator of the indices or the node/edge instances.

Returns (*generator*) – Generator object containing the node indices (or instances) having the specified attribute.

homology_translation(*target*, *source=None*, *only_swissprot=True*, *graph=None*)
Translates the current object to another organism by orthology. Proteins without known ortholog will be deleted.

Parameters **target** (*int*) – NCBI Taxonomy ID of the target organism. E.g. 10090 for mouse.

htp_stats()

in_complex(*csources=['corum']*)
Deprecated, will be removed.

in_directed(*vertex*)

in_undirected(*vertex*)

info(*name*)
Given the name of a resource, prints out the information about that source/database. You can check the list of available resource descriptions in `ypath.descriptions.descriptions.keys()`.

Parameters **name** (*str*) – The name of the resource from which to print the information.

init_complex_attr(*graph*, *name*)

init_edge_attr(*attr*)
Fills all edges attribute *attr* with its default type (if such attribute value is `None`), creates [list] if in `pypath.main.PyPath.edgeAttrs` such attribute is registered as [list].

Parameters **attr** (*str*) – The attribute name to be initialized on the network edges.

init_gsea(*user*)
Initializes a `pypath.gsea.GSEA` object and shows the list of the collections in MSigDB.

init_network(*lst=None*, *exclude=[]*, *cache_files={}*, *pickle_file=None*, *pfile=False*, *save=False*, *reread=None*, *redownload=None*, *keep_raw=False*, ***kwargs*)
Loads the network data.

This is a lazy way to start the module, load data and build the high confidence, literature curated part of the signaling network.

Parameters

- **lst** (*dict*) – Optional, `None` by default. Specifies the data input formats for the different resources (keys) [str]. Values are `pypath.input_formats.NetworkInput` instances containing the information. By default uses the set of resources of OmniPath.

- **exclude** (*list*) – Optional, `[]` by default. List of resources [str] to exclude from the network.
- **cache_files** (*dict*) – Optional, `{}` by default. Contains the resource name(s) [str] (keys) and the corresponding cached file name [str]. If provided (and file exists) bypasses the download of the data for that resource and uses the cache file instead.
- **pfile** (*str*) – Optional, `False` by default. If any, provides the file name or path to a previously saved network pickle file. If `True` is passed, takes the default path from *PyPath.save_network()* (`'cache/default_network.pickle'`).
- **save** (*bool*) – Optional, `False` by default. If set to `True`, saves the loaded network to its default location (`'cache/default_network.pickle'`).
- **reread** (*bool*) – Optional, `False` by default. Specifies whether to reread the data files from the cache or omit them (similar to *redownload*).
- **redownload** (*bool*) – Optional, `False` by default. Specifies whether to re-download the data and ignore the cache.
- ****kwargs** – Not used.

init_vertex_attr(*attr*)

Fills all vertices attribute *attr* with its default type (if such attribute value is `None`), creates [list] if in `pypath.main.PyPath.vertexAttrs` such attribute is registered as [list].

Parameters **attr** (*str*) – The attribute name to be initialized on the network vertices.

interactions_all(*resources=None*, ***kwargs*)

Returns a *set* of tuples of node name pairs representing interactions, both directed and undirected. Directed interactions will be present according to their direction, mutual directed interactions are represented by two tuples. The directed and undirected interactions are not distinguished here.

interactions_directed(*resources=None*, ***kwargs*)

Returns a *set* of tuples of node name pairs with being aware of the directions. Undirected interactions will be discarded. Pairs of node names represent the directions: first is the source, second is the target.

interactions_directed_by_resource(*resources=None*, *effect=None*, ***kwargs*)

Returns a *dict* of *set*s of tuples of node name pairs with being aware of the directions. Undirected interactions will be discarded. Pairs of node names represent the directions: first is the source, second is the target.

interactions_inhibitory(*resources=None*, ***kwargs*)

Returns a *set* of tuples of node name pairs only for inhibitory interactions. Pairs of node names represent the directions: first is the source, second is the target.

interactions_inhibitory_by_resource(*resources=None*, ***kwargs*)

Returns a *dict* of *set*s of tuples of node name pairs with being aware of the directions. Undirected interactions will be discarded. Pairs of node names represent the directions: first is the source, second is the target.

interactions_mutual(*resources=None*, ***kwargs*)

Returns a *set* of tuples of node name pairs representing mutual interactions (i.e. A–>B and B–>A). Pairs of node names will be sorted alphabetically.

interactions_mutual_by_resource(*resources=None*, ***kwargs*)

Returns a *dict* of *set*s of tuples of node name pairs representing mutual interactions (i.e. A–>B and B–>A). Pairs of node names will be sorted alphabetically.

interactions_signed(*resources=None*, ***kwargs*)

Returns a *set* of tuples of node name pairs only for signed interactions. Pairs of node names represent the directions: first is the source, second is the target.

interactions_signed_by_resource(*resources=None*, ***kwargs*)
Returns a *dict* of *set*s of tuples of node name pairs with being aware of the directions. Undirected interactions will be discarded. Pairs of node names represent the directions: first is the source, second is the target.

interactions_stimulatory(*resources=None*, ***kwargs*)
Returns a *set* of tuples of node name pairs only for stimulatory interactions. Pairs of node names represent the directions: first is the source, second is the target.

interactions_stimulatory_by_resource(*resources=None*, ***kwargs*)
Returns a *dict* of *set*s of tuples of node name pairs with being aware of the directions. Undirected interactions will be discarded. Pairs of node names represent the directions: first is the source, second is the target.

interactions_undirected(*resources=None*, ***kwargs*)
Returns a *set* of tuples of node name pairs without being aware of the directions. Pairs of node names will be sorted alphabetically.

interactions_undirected_by_resource(*resources=None*, ***kwargs*)
Returns a *dict* of *set*s of tuples of node name pairs without being aware of the directions. Pairs of node names will be sorted alphabetically.

intergroup_shortest_paths(*groupA*, *groupB*, *random=False*)

intogen_cancer_drivers_list(*intogen_file*)
Loads the list of cancer driver proteins from IntOGen data.

Parameters **intogen_file** (*str*) – Path to the data file. Can also be [function] that provides the data. In general, anything accepted by `pypath.input_formats.NetworkInput.input`.

iter_edges(*resources=None*)
Iterates the edges in the graph optionally limited to certain resources. Yields `igraph.Edge` objects.

iter_interactions(*signs=True*, *all_undirected=True*, *by_source=False*, *with_references=False*)
Iterates over edges and yields interaction records.

Parameters

- **signs** (*bool*) – Ignoring signs if `False`. This way each directed interaction will yield a single record even if it's ambiguously labeled both positive and negative. The default behaviour is to yield two records in this case, one with positive and one with negative sign.
- **all_undirected** (*bool*) – Yield records for undirected interactions even if certain sources provide direction. If `False` only the directed records will be provided and the undirected sources and references will be added to the directed records.
- **by_source** (*bool*) – Yield separate records by resources. This way the node pairs will be redundant and you need to group later if you want unique interacting pairs. By default is `False` because for most applications unique interactions are preferred. If `False` the *refrences* field will still be present but with `None` values.
- **with_references** (*bool*) – Include the literature references. By default is `False` because you rarely need these and they increase the data size significantly.

jaccard_edges()
Computes the Jaccard similarity index between the sets of first neighbours of all node pairs. **NOTE:** this method can take a while to compute, e.g.: if the network has 10K nodes, the total number of possible pairs to compute is:

$$\binom{10^4}{2} = 49995000$$

Returns (*list*) – Large list of [tuple] elements containing the node pair names [str] and their corresponding first neighbours Jaccard index [float].

jaccard_meta(*jedges*, *critical*)
Creates a (undirected) graph from a list of edges filtering by their Jaccard index.

Parameters

- **jedges** (*list*) – List of [tuple] containing the edges node names [str] and their Jaccard index. Basically, the output of `pypath.main.PyPath.jaccard_edges()`.
- **critical** (*float*) – Specifies the threshold of the Jaccard index from above which an edge will be included in the graph.

Returns (*igraph.Graph*) – The Undirected graph instance containing only the edges whose Jaccard similarity index is above the threshold specified by *critical*.

kegg_directions(*graph=None*)

kegg_pathways(*graph=None*)

kinase_stats()

label(*label*, *idx*, *what='vertices'*)
Creates a boolean attribute `label` True for the vertex or edge IDs in the set `idx`.

label_by_go(*label*, *go_terms*, *method='ANY'*)
Assigns a boolean vertex attribute to nodes which tells whether the node is annotated by all or any of the GO terms.

label_edges(*label*, *edges*)
Creates a boolean edge attribute `label` True for the edge IDs in the set `edges`.

label_vertices(*label*, *vertices*)
Creates a boolean vertex attribute `label` True for the vertex IDs in the set `vertices`.

laudanna_directions(*graph=None*)

laudanna_effects(*graph=None*)

static license(*self*)
Prints information about data licences.

static list_resources()
Prints the list of resources through the standard output.

load_3dcomplexes(*graph=None*)

load_3did_ddi()

load_3did_ddi2(*ddi=True*, *interfaces=False*)

load_3did_dmi()

load_3did_interfaces()

load_all_pathways(*graph=None*)

load_compleat(*graph=None*)
Loads complexes from Compleat. Loads data into vertex attribute *graph.vs['complexes']['compleat']*. This resource is human only.

load_complexportal(*graph=None*)
Loads complexes from ComplexPortal. Loads data into vertex attribute *graph.vs['complexes']['complexportal']*. This resource is human only.

load_comppi(*graph=None*)

load_corum(*graph=None*)
Loads complexes from CORUM database. Loads data into vertex attribute *graph.vs['complexes']['corum']*. This resource is human only.

load_dbptm(*non_matching=False*, *trace=False*, ***kwargs*)

load_ddi(*ddi*)
ddi is either a list of intera.DomainDomain objects, or a function resulting this list

load_ddis(*methods=['dataio.get_3dc_ddi', 'dataio.get_domino_ddi', 'self.load_3did_ddi2']*)

load_depod_dmi()

load_disgenet(*dataset='curated'*, *score=0.0*, *umls=False*, *full_data=False*)
Assigns DisGeNet disease-gene associations to the proteins in the network. Disease annotations will be added to the *dis* vertex attribute.

Parameters

- **score** (*float*) – Confidence score from DisGeNet. Only associations above the score provided will be considered.
- **ulms** (*bool*) – By default we assign a list of disease names to each protein. To use Unified Medical Language System IDs instead set this to *True*.
- **full_data** (*bool*) – By default we load only disease names. Set this to *True* if you wish to load additional annotations like number of PubMed IDs, number of SNPs and original sources.

load_dmi(*dmi*)
dmi is either a list of intera.DomainMotif objects, or a function resulting this list

load_dmis(*methods=['self.pfam_regions', 'self.load_depod_dmi', 'self.load_dbptm', 'self.load_mimp_dmi', 'self.load_pnetworks_dmi', 'self.load_domino_dmi', 'self.load_pepcyber', 'self.load_psite_reg', 'self.load_psite_phos', 'self.load_ielm', 'self.load_phosphoelm', 'self.load_elm', 'self.load_3did_dmi']*)

load_domino_dmi(*organism=None*)

load_dorothea(*levels={'A', 'B'}*, *only_curated=False*)
Adds TF-target interactions from TF regulons to the network. DoRothEA is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

Parameters

- **levels** (*set*) – Optional, `{'A', 'B'}` by default. Confidence levels to be loaded (from A to E) [str].
- **only_curated** (*bool*) – Optional, `False` by default. Whether to retrieve only the literature curated interactions or not.

load_elm()

load_exocarta_attrs(*load_samples=False*, *load_refs=False*)
Creates vertex attributes from ExoCarta data. Creates a boolean attribute `exocarts_exosomal` which tells whether a protein is in ExoCarta i.e. has been found in exosomes. Optionally creates attributes `exocarta_samples` and `exocarta_refs` listing the sample tissue and the PubMed references, respectively.

load_expression(*array=False*)
Expression data can be loaded into vertex attributes, or into a pandas DataFrame – the latter offers faster ways to process and use these huge matrices.

load_from_pickle(*pickle_file*)
Shortcut for loading a network from a pickle dump.

load_go(*organism=None*)
Creates a `pypath.go.GOAnnotation` object for one organism in the dict under `go` attribute.

Parameters **organism** (*int*) – NCBI Taxonomy ID of the organism.

load_havugimana(*graph=None*)
Loads complexes from Havugimana 2012. Loads data into vertex attribute *graph.vs['complexes']['havugimana']*. This resource is human only.

load_hpa(*normal=True*, *pathology=True*, *cancer=True*, *summarize_pathology=True*, *tissues=None*, *quality={'Approved'*, *'Supported'}*, *levels={'High': 3*, *'Low': 1*, *'Medium': 2*, *'Not detected': 0}*, *graph=None*, *na_value=0*)
Loads Human Protein Atlas data into vertex attributes.

load_hprd_ptms(*non_matching=False*, *trace=False*, ***kwargs*)

load_ielm()

load_interfaces()

load_li2012_ptms(*non_matching=False*, *trace=False*, ***kwargs*)

load_ligand_receptor_network(*lig_rec_resources=True*, *inference_from_go=True*, *sources=None*, *keep_undirected=False*, *keep_rec_rec=False*, *keep_lig_lig=False*)
Initializes a ligand-receptor network.

load_lmpid(*method*)

load_matrisome_attrs(*organism=None*)
Loads vertex attributes from MatrisomeDB 2.0. Attributes are `matrisome_class`, `matrisome_subclass` and `matrisome_notes`.

load_membranome_attrs()
Loads attributes from Membranome, a database of single-helix transmembrane proteins.

load_mimp_dmi(*non_matching=False*, *trace=False*, ***kwargs*)

load_mutations(*attributes=None*, *gdsc_datadir=None*, *mutation_file=None*)
Mutations are listed in vertex attributes. Mutation() objects offers methods to identify residues and look up in Ptm(), Motif() and Domain() objects, to check if those residues are modified, or are in some short motif or domain.

load_negatives()

load_old_omnipath(*kinase_substrate_extra=False*, *remove_htp=False*, *htp_threshold=1*, *keep_directed=False*, *min_refs_undirected=2*)
Loads the OmniPath network as it was before August 2016. Furthermore it gives some more options.

load_omnipath(*omnipath=None*, *kinase_substrate_extra=False*, *ligand_receptor_extra=False*, *pathway_extra=False*, *remove_htp=True*, *htp_threshold=1*, *keep_directed=True*, *min_refs_undirected=2*, *old_omnipath_resources=False*, *exclude=None*, *pickle_file=None*)
Loads the OmniPath network. Note, if `pickle_file` provided the network will be loaded directly from there regardless of its content.

load_pathways(*source*, *graph=None*)
Generic method to load pathway annotations from a resource. We don't recommend calling this method but either specific methods for a single source e.g. *kegg_pathways()* or *sirnor_pathways()* or call *load_all_pathways()* to load all resources.

Parameters

- **source** (*str*) – Name of the source, this need to match a method in the dict in *get_pathways()* method and the edge and vertex attributes with pathway annotations will be called "<source>_pathways".
- **graph** (*igraph.Graph*) – A graph, by default the default the *graph* attribute of the current instance.

load_pdb(*graph=None*)
Loads the 3D structure information from PDB into the network. Creates the node attribute `'pdb'` containing a [dict] whose keys are the PDB identifier [str] and values are [tuple] of two elements denoting the experimental method [str] (e.g.: `'X-ray'`, `'NMR'`, ...) and the resolution [float] (if applicable).

Parameters **graph** (*igraph.Graph*) – Optional, `None` by default. The network object for which the information is to be loaded. If none is passed, takes the undirected network of the current instance.

load_pepcyber()

load_pfam(*graph=None*)
Loads the protein family information from UniProt into the network. Creates the node attribute `'pfam'` containing a [list] of protein family identifier(s) [str].

Parameters **graph** (*igraph.Graph*) – Optional, `None` by default. The network object for which the information is to be loaded. If none is passed, takes the undirected network of the current instance.

load_pfam2()
Loads the protein family information from Pfam into the network. Creates the node attribute `'pfam'` containing a [list] of [dict] whose keys are protein family identifier(s) [str] and corresponding values are [list] of [dict] containing detailed information about the protein family(ies) for regions and isoforms of the protein.

Parameters **graph** (*igraph.Graph*) – Optional, `None` by default. The network object for which the information is to be loaded. If none is passed, takes the undirected network of the current instance.

load_pfam3()
Loads the protein domain information from Pfam into the network. Creates the node attribute `'doms'` containing a [list] of `pypath.intera.Domain` instances with information about each domain of the protein (see the corresponding class documentation for more information).

load_phospho_dmi(*source*, *trace=False*, *return_raw=False*, ***kwargs*)

load_phosphoelm(*trace=False*, ***kwargs*)

load_pisa(*graph=None*)

load_pnetworks_dmi(*trace=False*, ***kwargs*)

load_psite_phos(*trace=False*, ***kwargs*)

load_psite_reg()

load_ptms()

load_ptms2(*input_methods=None, map_by_homology_from=[9606], homology_only_swissprot=True, ptm_homology_strict=False, nonhuman_direct_lookup=True, inputargs={}, database=None, force_load=False*)

This is a new method which will replace *load_ptms*. It uses *pypath.enz_sub.EnzymeSubstrateAggregator*, a newly introduced module for combining enzyme-substrate data from multiple resources using homology translation on users demand.

Parameters

- **input_methods** (*list*) – Resources to collect enzyme-substrate interactions from. E.g. *['Signor', 'phosphoELM']*. By default it contains Signor, PhosphoSitePlus, HPRD, phosphoELM, dbPTM, PhosphoNetworks, Li2012 and MIMP.
- **map_by_homology_from** (*list*) – List of NCBI Taxonomy IDs of source taxons used for homology translation of enzyme-substrate interactions. If you have a human network and you add here *[10090, 10116]* then mouse and rat interactions from the source databases will be translated to human.
- **homology_only_swissprot** (*bool*) – *True* by default which means only SwissProt IDs are accepted at homology translateion, Trembl IDs will be dropped.
- **ptm_homology_strict** (*bool*) – For homology translation use PhosphoSite's PTM homology table. This guarantees that only truely homologous sites will be included. Otherwise we only check if at the same numeric offset in the homologous sequence the appropriate residue can be find.
- **nonhuman_direct_lookup** (*bool*) – Fetch also directly nonhuman data from the resources whereever it's available. PhosphoSite contains mouse enzyme-substrate interactions and it is possible to extract these directly beside translating the human ones to mouse.
- **inputargs** (*dict*) – Additional arguments passed to *PtmProcessor*. A *dict* can be supplied for each resource, e.g. *{'Signor': {...}, 'PhosphoSite': {...}, ...}*. Those not used by *PtmProcessor* are forwarded to the *pypath.dataio* methods.
- **database** – A `PtmAggregator` object. If provided no new database will be created.
- **force_load** (*bool*) – If `True` the database will be loaded with the parameters provided here; otherwise if the `ptm` module already has a database no new database will be created. This means the parameters specified in other arguments might have no effect.

load_resource(*settings, clean=True, cache_files={}, reread=None, redownload=None, keep_raw=False*)

Loads the data from a single resource and attaches it to the network

Parameters

- **settings** (*pypath.input_formats.NetworkInput*) – `pypath.input_formats.NetworkInput` instance containing the detailed definition of the input format to the downloaded file.
- **clean** (*bool*) – Optional, `True` by default. Whether to clean the graph after importing the data or not. See `pypath.main.PyPath.clean_graph()` for more information.
- **cache_files** (*dict*) – Optional, `{}` by default. Contains the resource name(s) [str] (keys) and the corresponding cached file name [str]. If provided (and file exists) bypasses the download of the data for that resource and uses the cache file instead.
- **reread** (*bool*) – Optional, `False` by default. Specifies whether to reread the data files from the cache or omit them (similar to *redownload*).

- **redownload** (*bool*) – Optional, `False` by default. Specifies whether to re-download the data and ignore the cache.

load_resources(*lst=None*, *exclude=[]*, *cache_files={}*, *reread=False*, *redownload=None*, *keep_raw=False*)

Loads multiple resources, and cleans up after. Looks up ID types, and loads all ID conversion tables from UniProt if necessary. This is much faster than loading the ID conversion and the resources one by one.

Parameters

- **lst** (*dict*) – Optional, `None` by default. Specifies the data input formats for the different resources (keys) [str]. Values are `pypath.input_formats.NetworkInput` instances containing the information. By default uses the set of resources of OmniPath.
- **exclude** (*list*) – Optional, `[]` by default. List of resources [str] to exclude from the network.
- **cache_files** (*dict*) – Optional, `{}` by default. Contains the resource name(s) [str] (keys) and the corresponding cached file name [str]. If provided (and file exists) bypasses the download of the data for that resource and uses the cache file instead.
- **reread** (*bool*) – Optional, `False` by default. Specifies whether to reread the data files from the cache or omit them (similar to *redownload*).
- **redownload** (*bool*) – Optional, `False` by default. Specifies whether to re-download the data and ignore the cache.

load_signor_ptms(*non_matching=False*, *trace=False*, ***kwargs*)

load_surfaceome_attrs()

Loads vertex attributes from the In Silico Human Surfaceome. Attributes are `surfaceome_score`, `surfaceome_class` and `surfaceome_subclass`.

load_tfregulons(*levels={'A', 'B'}*, *only_curated=False*)

Adds TF-target interactions from TF regulons to the network. DoRothEA is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

Parameters

- **levels** (*set*) – Optional, `{'A', 'B'}` by default. Confidence levels to be loaded (from A to E) [str].
- **only_curated** (*bool*) – Optional, `False` by default. Whether to retrieve only the literature curated interactions or not.

load_vesiclepedia_attrs(*load_samples=False*, *load_refs=False*, *load_vesicle_type=False*)

Creates vertex attributes from Vesiclepedia data. Creates a boolean attribute `vesiclepedia_in_vesicle` which tells whether a protein is in ExoCarta i.e. has been found in exosomes. Optionally creates attributes `vesiclepedia_samples`, `vesiclepedia_refs` and `vesiclepedia_vesicles` listing the sample tissue, the PubMed references and the vesicle types, respectively.

lookup_cache(*name*, *cache_files*, *int_cache*, *edges_cache*)

Checks up the cache folder for the files of a given resource. First checks if *name* is on the *cache_files* dictionary. If so, loads either the interactions or edges otherwise. If not, checks *edges_cache* or *int_cache* otherwise.

Parameters

- **name** (*str*) – Name of the resource (lower-case).
- **cache_files** (*dict*) – Contains the resource name(s) [str] (keys) and the corresponding cached file name [str] (values).
- **int_cache** (*str*) – Path to the interactions cache file of the resource.
- **edges_cache** (*str*) – Path to the edges cache file of the resource.

Returns

- (*file*) – The loaded pickle file from the cache if the file is contains the interactions. `None` otherwise.
- (*list*) – List of mapped edges if the file contains the information from the edges. `[]` otherwise.

loop_edges(*index=True*, *graph=None*)

Returns an iterator of the indices (or the edge instances) of the edges which represent a loop (whose source and target node are the same).

Parameters

- **index** (*bool*) – Optional, `True` by default. Whether to return the iterator of the indices or the edge instances.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object where the edge loops are to be searched. If none is passed, takes the undirected network of the current instance.

Returns (*generator*) – Generator object containing the edge indices (or instances) containing loops.

mean_reference_per_interaction(*resources=None*)

Computes the mean number of references per interaction of the network.

Returns (*float*) – Mean number of interactions per edge.

mean_reference_per_interaction_by_resource(*resources=None*)

Computes the mean number of references per interaction of the network.

Returns (*float*) – Mean number of interactions per edge.

merge_lists(*id_a*, *id_b*, *name=None*, *and_or='and'*, *delete=False*, *func='max'*)

Merges two lists from `pypat.main.PyPath.lists`.

Parameters

- **id_a** (*str*) – Name of the first list to be merged.
- **id_b** (*str*) – Name of the second list to be merged.
- **name** (*str*) – Optional, `None` by default. Specifies a new name for the merged list. If none is passed, name will be set to *id_a*_*id_b*.
- **and_or** (*str*) – Optional, `'and'` by default. The logic operation perfomed in the merging: `'and'` performs an union, `'or'` for the intersection.
- **delete** (*bool*) – Optional, `False` by default. Whether to delete the former lists or not.
- **func** (*str*) – Optional, `'max'` by default. Not used.

merge_nodes(*nodes*, *primary=None*, *graph=None*)

Merges all attributes and edges of selected nodes and assigns them to the primary node (by default the one with lowest index).

Parameters

- **nodes** (*list*) – List of node indexes [int] that are to be collapsed.
- **primary** (*int*) – Optional, `None` by default. ID of the primary edge, if none is passed, the node with lowest index on *nodes* is selected.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The network graph object from which the nodes are to be merged. If none is passed, takes the undirected network graph.

mimp_directions(*graph=None*)

mutated_edges(*sample*)
Compares the mutated residues and the modified residues in PTMs. Interactions are marked as mutated if the target residue in the underlying PTM is mutated.

name_edgelist(*graph=None*)
Returns an edge list, i.e. a list with tuples of vertex names.

names2vids(*names*)
From a list of node names, returns their corresponding indices.

Parameters **names** (*list*) – Contains the node names [str] for which the IDs are to be searched.

Returns (*list*) – The queried node IDs [int].

negative_report(*lst=True*, *outFile=None*)
Generates a report file with the negative interactions (assumed to be already loaded).

Parameters

- **lst** (*bool*) – Optional, `True` by default. Whether to retun a list of edges containing the edge instances which have negative references.
- **outFile** (*str*) – Optional, `None` by default. The output file name/path. If none is passed, the default is `'results/<session_id>-negatives'`

Returns (*list*) – If *lst* is set to `True`, returns a [list] is returned with the `igraph.Edge` instances that contain at least a negative reference.

neighborhood(*identifiers*, *order=1*, *mode='ALL'*)

neighbors(*identifier*, *mode='ALL'*)

neighbourhood_network(*center*, *second=False*)

network_by_go(*node_categories*, *network_sources=None*, *include=None*, *exclude=None*, *directed=False*, *keep_undirected=False*, *prefix='GO'*, *delete=True*, *copy=False*, *vertex_attrs=True*, *edge_attrs=True*)
Creates or filters a network based on Gene Ontology annotations.

Parameters

- **node_categories** (*dict*) – A dict with custom category labels as keys and expressions of GO terms as values. E.g. ‘‘{‘extracell’: ‘GO:0005576 and not GO:0070062’,

  ’plasmamem’: ‘GO:0005887’}‘‘.
- **network_sources** (*dict*) – A dict with anything as keys and network input format definintions (`input_formats.NetworkInput` instances) as values.
- **include** (*list*) – A list of tuples of category label pairs. By default we keep all edges connecting proteins annotated with any of the defined categories. If `include` is defined then only edges between category pairs defined here will be kept and all others deleted.

- **exclude** (*list*) – Similarly to include, all edges will be kept but the ones listed in `exclude` will be deleted.
- **directed** (*bool*) – If True `include` and `exclude` relations will be processed with directed (source, target) else direction won't be considered.
- **keep_undirected** (*bool*) – If True the interactions without direction information will be kept even if `directed` is True. Passed to `edges_between` as `strict` argument.
- **prefix** (*str*) – Prefix for all vertex and edge attributes created in this operation. E.g. if you have a category label 'bar' and prefix is 'foo' then you will have a new vertex attribute 'foo__bar'.
- **delete** (*bool*) – Delete the vertices and edges which don't belong to any of the categories.
- **copy** (*bool*) – Return a copy of the entire `PyPath` object with the graph filtered by GO terms. By default the object is modified in place and `None` is returned.
- **vertex_attrs** (*bool*) – Create vertex attributes.
- **edge_attrs** (*bool*) – Create edge attributes.

network_filter (*p=2.0*)
This function aims to cut the number of edges in the network, without losing nodes, to make the network less connected, less hairball-like, more usable for analysis.

network_stats (*outfile=None*)
Calculates basic statistics for the whole network and each of sources (node and edge counts, average node degree, graph diameter, transitivity, adhesion and cohesion). Writes the results in a tab file. File is stored in `pypath.main.PyPath.outdir` (`'results'` by default).

Parameters **outfile** (*str*) – Optional, `None` by default. Specifies the file name. If none is specified, this will be `'pwnet-<session_id>-stats'`.

new_edges (*edges*)
Adds new edges from any iterable of edges to the undirected graph. Basically, calls `igraph.Graph.add_edges()`.

Parameters **edges** (*list*) – Contains the edges that are to be added to the network.

new_nodes (*nodes*)
Adds new nodes from any iterable of nodes to the undirected graph. Basically, calls `igraph.Graph.add_vertices()`.

Parameters **nodes** (*list*) – Contains the nodes that are to be added to the network.

node_exists (*name*)
Checks if a node exists in the (undirected) network.

Parameters **name** (*str*) – The name of the node to be searched.

Returns (*bool*) – Whether the node exists in the network or not.

numof_directed_edges ()

numof_edges (*resources=None*)
Number of edges optionally limited to certain resources.

numof_reference_interaction_pairs ()
Returns the total of unique references per interaction.

Returns (*int*) – Total number of unique references per interaction.

numof_references_by_resource(*resources=None*, ***kwargs*)
Counts the references for each resource, optionally limited to certain resources.

numof_undirected_edges()

orthology_translation(*target*, *source=None*, *only_swissprot=True*, *graph=None*)
Translates the current object to another organism by orthology. Proteins without known ortholog will be deleted.

Parameters **target** (*int*) – NCBI Taxonomy ID of the target organism. E.g. 10090 for mouse.

p(*identifier*)
Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default undirected graph, or a UniProt ID or GeneSymbol which can be found in the default undirected network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

pathway_attributes(*graph=None*)

pathway_members(*pathway*, *source*)
Returns an iterator with the members of a single pathway. Apart from the pathway name you need to supply its source database too.

pathway_names(*source*, *graph=None*)
Returns the names of all pathways having at least one member in the current graph.

pathway_similarity(*outfile=None*)
Computes the Sorensen's similarity index across nodes and edges for all the available pathway sources (already loaded in the network) and saves them into table files. Files are stored in `pypath.main.PyPath.outdir` (`'results'` by default). See `pypath.main.PyPath.sorensen_pathways()` for more information..

Parameters **outfile** (*str*) – Optional, `None` by default. Specifies the file name prefix (suffixes will be `'-nodes'` and `'-edges'`). If none is specified, this will be `'pwnet-<session_id>-sim-pw'`.

pathways_table(*filename='genes_pathways.list', pw_sources=['signalink', 'signor', 'netpath', 'kegg'], graph=None*)

pfam_regions()

phosphonetworks_directions(*graph=None*)

phosphopoint_directions(*graph=None*)

phosphorylation_directions()

phosphorylation_signs()

phosphosite_directions(*graph=None*)

prdb_tissue_expr(*tissue*, *prdb=None*, *graph=None*, *occurrence=1*, *group_function=<function PyPath.<lambda>>*, *na_value=0.0*)

process_directions(*dirs*, *name*, *directed=None*, *stimulation=None*, *inhibition=None*, *graph=None*, *id_type=None*, *dirs_only=False*)

process_dmi(*source*, ***kwargs*)
This is an universal function for loading domain-motif objects like load_phospho_dmi() for PTMs. TODO this will replace load_elm, load_ielm, etc

protein(*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the

identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

protein_edge (*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

Parameters

- **source** (*int, str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **target** (*int, str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **directed** (*bool*) – To be passed to igraph.Graph.get_eid()

proteins (*identifiers*)

ps (*identifiers*)

random_walk_with_return (*q*, *graph=None*, *c=0.5*, *niter=1000*)
Random walk with return (RWR) starting from one or more query nodes. Returns affinity (probability) vector of all nodes in the graph.

param int,list q Vertex IDs of query nodes.

param igraph.Graph graph An *igraph.Graph* object.

param float c Probability of restart.

param int niter Number of iterations.

```
>>> import igraph
>>> import pypath
>>> pa = pypath.PyPath()
>>> pa.init_network({
        'signor': pypath.data_formats.pathway['signor']
    })
>>> q = [
        pa.gs('EGFR').index,
        pa.gs('ATG4B').index
    ]
>>> rwr = pa.random_walk_with_return(q = q)
>>> palette = igraph.RainbowPalette(n = 100)
>>> colors  = [palette.get(int(round(i))) for i in rwr / max(rwr) * 99]
>>> igraph.plot(pa.graph, vertex_color = colors)
```

random_walk_with_return2 (*q*, *c=0.5*, *niter=1000*)
Literally does random walks. Only for testing of the other method, to be deleted later.

read_from_cache (*cache_file*)
Reads a pickle file from the cache and returns it. It is assumed that the subfolder `cache/` is on the supplied path.

Parameters **cache_file** (*str*) – Path to the cache file that is to be loaded.

Returns (*file*) – The loaded pickle file from the cache. Type will depend on the file itself (e.g.: if the pickle was saved from a dictionary, the type will be [dict]).

read_list_file (*settings*, ***kwargs*)
Reads a list from a file and adds it to `pypath.main.PyPath.lists`.

Parameters

- **settings** (*pypath.input_formats.ReadList*) – python.data_formats.ReadList instance specifying the settings of the file to be read. See the class documentation for more details.
- ****kwargs** – Extra arguments passed to the file reading function. Such function name is outlined in the python.data_formats.ReadList.input attribute and defined in pypath.dataio.

reference_edge_ratio()
Computes the average number of references per edge (as in the undirected graph).

Returns (*float*) – Average number of references per edge.

reference_hist(*filename=None*)
Generates a file containing a table with information about the network's edges. First column contains the source node ID, followed by the target's ID, third column contains the number of references for that interaction and finally the number of sources. Writes the results in a tab file.

Parameters **filename** (*str*) – Optional, None by default. Specifies the file name and path to save the table. If none is passed, file will be saved in pypath.main.PyPath.outdir ('results' by default) with the name '<session_id>-refs-hist'.

references(*resources=None*, ***kwargs*)
Returns a set of references for all edges.

resources [None,str,set] Limits the query to one or more resources.

references_by_resource(*resources=None*, ***kwargs*)
Creates a dict with resources as keys and sets of references as values.

reload()
Reloads the object from the module level.

remove_htp(*threshold=50*, *keep_directed=False*)

remove_undirected(*min_refs=None*)

resources
All network resources. Returns *set* of strings.

run_batch(*methods*, *toCall=None*)

save_network(*pickle_file=None*, *pfile=None*)
Saves the network object.

Stores the instance into a pickle (binary) file which can be reloaded in the future.

Parameters **pickle_file** (*str*) – Optional, None by default. The path/file name where to store the pcikle file. If not specified, saves the network to its default location ('cache/default_network.pickle').

save_session()
Save the current session state into pickle dump. The file will be saved in the current working directory as 'pypath-<session_id>.pickle'.

save_to_pickle(*pickle_file=None*, *pfile=None*)
Saves the network object.

Stores the instance into a pickle (binary) file which can be reloaded in the future.

Parameters **pickle_file** (*str*) – Optional, `None` by default. The path/file name where to store the pcikle file. If not specified, saves the network to its default location (`'cache/default_network.pickle'`).

search_attr_and(*obj*, *lst*)
Searches a given collection of attributes in a given object. Only returns `True`, if all elements of *lst* can be found in *obj*.

Parameters

- **obj** (*object*) – Object (dictionary-like) where to search for elements of *lst*.
- **lst** (*dict*) – Keys are the attribute names [str] and values the collection of elements to be searched in such attribute [set].

Returns (*bool*) – `True` only if *lst* is empty or all of its elements are found in *obj*. Returns `False` otherwise (as soon as one element of *lst* is not found).

search_attr_or(*obj*, *lst*)
Searches a given collection of attributes in a given object. As soon as one item is found, returns `True`, if none could be found then returns `False`.

Parameters

- **obj** (*object*) – Object (dictionary-like) where to search for elements of *lst*.
- **lst** (*dict*) – Keys are the attribute names [str] and values the collection of elements to be searched in such attribute [set].

Returns (*bool*) – `True` if *lst* is empty or any of its elements is found in *obj*. Returns only `False` if cannot find anything.

second_neighbours(*node*, *indices=False*, *with_first=False*)
Looks for the (first and) second neighbours of a given node and returns a list of their UniProt IDs.

Parameters

- **node** (*str*) – The UniProt ID of the node of interest. Can also be the index of such node [int].
- **indices** (*bool*) – Optional, `False` by default. Whether to return the neighbour nodes indices or their UniProt IDs.
- **wit_first** (*bool*) – Optional, `False` by default. Whether to return also the first neighbours or not.

Returns (*list*) – The list containing the second neighbours of the queried node (including the first ones if specified).

select_by_go(*go_terms*)
Retrieves the vertex IDs of all vertices annotated with any Gene Ontology terms or their descendants, or evaluates string expression (see `select_by_go_expr`).

Parameters **go_terms** (*str,set*) – A single GO term, a set of GO terms or an expression with GO terms.

select_by_go_all(*go_terms*)
Selects the nodes annotated by all GO terms in `go_terms`.

Returns set of vertex IDs.

Parameters **go_terms** (*list*) – List, set or tuple of GO terms.

select_by_go_expr(*go_expr*)
Selects vertices based on an expression of Gene Ontology terms. Operator precedence not considered, please use parentheses.

Parameters **go_expr** (*str*) – An expression of Gene Ontology terms. E.g. `'(GO:0005576 and not GO:0070062) or GO:0005887'`. Parentheses and operators `and`, `or` and `not` can be used.

separate()
Separates the undirected network according to the different sources. Basically applies `pypath.main.PyPath.get_network()` for each resource.

Returns (*dict*) – Keys are resource names [str] whose values are the subnetwork [igraph.Graph] containing the elements of that source.

separate_by_category()
Separates the undirected network according to resource categories. Possible categories are:

- `'m'`: PTM/enzyme-substrate resources.
- `'p'`: Pathway/activity flow resources.
- `'i'`: Undirected/PPI resources.
- `'r'`: Process description/reaction resources.
- `'t'`: Transcription resources.

Works in the same way as `pypath.main.PyPath.separate()`.

Returns (*dict*) – Keys are category names [str] whose values are the subnetwork [igraph.Graph] containing the elements of those resources corresponding to that category.

sequences(*isoforms=True*, *update=False*)

set_boolean_vattr(*attr*, *vids*, *negate=False*)

set_categories()
Sets the category attribute on the network nodes and edges (`'cat'`) as well the edge attribute coercing the references by category (`'refs_by_cat'`). The possible categories are as follows:

- `'m'`: PTM/enzyme-substrate resources.
- `'p'`: Pathway/activity flow resources.
- `'i'`: Undirected/PPI resources.
- `'r'`: Process description/reaction resources.
- `'t'`: Transcription resources.

set_chembl_mysql(*title*, *config_file=None*)
Sets the ChEMBL MySQL configuration according to the *title* section in *config_file* ini file configuration.

Parameters

- **title** (*str*) – Section title of the ini file.
- **config_file** (*str*) – Optional, `None` by default. Specifies the configuration file name if none is passed, `mysql_config/defaults.mysql` will be used.

set_disease_genes(*dataset='curated'*)
Creates a vertex attribute named *dis* with boolean values *True* if the protein encoded by a disease related gene according to DisGeNet.

Parameters **dataset** (*str*) – Which dataset to use from DisGeNet. Default is *curated*.

set_druggability()
Creates a vertex attribute *dgb* with value *True* if the protein is druggable, otherwise *False*.

set_drugtargets(*pchembl=5.0*)
Creates a vertex attribute *dtg* with value *True* if the protein has at least one compound binding with affinity higher than *pchembl*, otherwise *False*.

Parameters **pchembl** (*float*) – Pchembl threshold.

set_kinases()
Creates a vertex attribute *kin* with value *True* if the protein is a kinase, otherwise *False*.

set_plasma_membrane_proteins_cspa()
Creates a vertex attribute *cspa* with value *True* if the protein is a plasma membrane protein according to CPSA, otherwise *False*.

set_plasma_membrane_proteins_cspa_surfaceome(*score_threshold=0.0*)
Creates a vertex attribute `surf` with value *True* if the protein is a plasma membrane protein according either to the Cell Surface Protein Atlas or the In Silico Human Surfaceome.

set_plasma_membrane_proteins_surfaceome(*score_threshold=0.0*)
Creates a vertex attribute *ishs* with value *True* if the protein is a plasma membrane protein according to the In Silico Human Surfaceome, otherwise *False*.

set_receptors()
Creates a vertex attribute *rec* with value *True* if the protein is a receptor, otherwise *False*.

set_signaling_proteins()
Creates a vertex attribute *kin* with value *True* if the protein is a kinase, otherwise *False*.

set_tfs(*classes=['a', 'b', 'other']*)

set_transcription_factors(*classes=['a', 'b', 'other']*)
Creates a vertex attribute *tf* with value *True* if the protein is a transcription factor, otherwise *False*.

Parameters **classes** (*list*) – Classes to use from TF Census. Default is *['a', 'b', 'other']*.

shortest_path_dist(*graph=None*, *subset=None*, *outfile=None*, ***kwargs*)
Computes the distribution of shortest paths for each pair of nodes in the network (or between group(s) of nodes if *subset* is provided). **NOTE:** this method can take a while to compute, e.g.: if the network has 10K nodes, the total number of possible pairs to compute is:

$$\binom{10^4}{2} = 49995000$$

Parameters

- **graph** (*igraph.Graph*) – Optional, `None` by default. The network object for which the shortest path distribution is to be computed. If none is passed, takes the undirected network of the current instance.
- **susbet** (*tuple*) – Optional, `None` by default. Contains two lists of node indices defining two groups between which the distribution is to be computed. Can also be [list] if the shortest paths are to be searched whithin the group. If none is passed, the whole network is taken by default.
- **outfile** (*str*) – Optional, `None` by default. File name/path to save the shortest path distribution. If none is passed, no file is generated.
- ****kwargs** – Additional keyword arguments passed to `igraph.Graph.get_shortest_paths()`.

Returns (*list*) – The length of the shortest paths for each pair of nodes of the network (or whithin/between group/s if *subset* is provided).

signaling_proteins_list()
Compiles a list of signaling proteins (as opposed to other proteins like metabolic enzymes, matrix proteins, etc), by looking up a few simple keywords in short description of GO terms.

signor_pathways(*graph=None*)

similarity_groups(*groups*, *index='simpson'*)
Computes the similarity index across the given *groups*.

Parameters

- **groups** (*dict*) – Contains the different group names [str] as keys and their corresponding elements [set].
- **index** (*str*) – Optional, `'simpson'` by default. The type of index metric to use to compute the similarity. Options are `'simpson'`, `'sorensen'` and `'jaccard'`.

Returns (*dict*) – Dictionary of dictionaries containing the groups names [str] as keys (for both inner and outer dictionaries) and the index metric as inner value [float] between those groups.

small_plot(*graph*, ***kwargs*)
This method is deprecated, do not use it.

sorensen_pathways(*pwlist=None*)
Computes the Sorensen's similarity index across nodes and edges for the given list of pathway sources (all loaded pathway sources by default).

Parameters **pwlist** (*list*) – Optional, `None` by default. The list of pathway sources to be compared.

Returns (*dict*) – Nested dictionaries (three levels). First-level keys are `'nodes'` and `'edges'`, then second and third levels correspond to `<source>__<patwhay>` names which map to the similarity index between those pathways [float].

source_diagram(*outf=None*, ***kwargs*)

source_network(*font='HelveticaNeueLTStd'*)
For EMBL branding, use Helvetica Neue Linotype Standard light

source_similarity(*outfile=None*)
Computes the Sorensen's similarity index across nodes and edges for all the sources available (already loaded in the network) and saves them into table files. Files are stored in `pypath.main.PyPath.outdir` (`'results'` by default). See `pypath.main.PyPath.databases_similarity()` for more information.

Parameters **outfile** (*str*) – Optional, `None` by default. Specifies the file name prefix (suffixes will be `'-nodes'` and `'-edges'`). If none is specified, this will be `'pwnet-<session_id>-sim-src'`.

source_stats()

sources_hist()
Counts the number of sources per interaction in the graph and saves them into a file named `source_num`. File is stored in `pypath.main.PyPath.outdir` (`'results'` by default).

sources_overlap(*diagonal=False*)

sources_venn_data(*fname=None*, *return_data=False*)
Computes the overlap in number of interactions for all pairs of sources.

Parameters

- **fname** (*str*) – Optional, `None` by default. If provided, saves the results into a table file. File is stored in `pypath.main.PyPath.outdir` (`'results'` by default).
- **return_data** (*bool*) – Optional, `False` by default. Whether to return the results as a [list].

Returns (*list*) – Only if *return_data* is set to `True`. List of lists containing the counts for each pair of resources. This is, for instance, number of interactions only in resource A, number of interactions only in resource B and number of common interactions between A and B.

stats (*method*, *keep_collection=False*, ***kwargs*)
Creates a collection of entities over the network according to `method` and counts them. By default the collection won't be returned but only the counts.

straight_between (*id_a*, *id_b*)
Finds an edge between the provided node names.

Parameters

- **id_a** (*str*) – The name of the source node.
- **id_b** (*str*) – The name of the target node.

Returns (*int*) – The edge ID. If the edge doesn't exist, returns [list] with the node indices [int].

string_effects (*graph=None*)

sum_in_complex (*csources=['corum'], graph=None*)
Returns the total number of edges in the network falling between two members of the same complex. Returns as a dict by complex resources. Calls :py:func:pypath.pypath.Pypath.edges_in_comlexes() to do the calculations.

@csources [list] List of complex resources. Should be already loaded.

@graph [igraph.Graph()] The graph object to do the calculations on.

summaries_tab (*outfile=None*, *return_table=False*)
Creates a table from resource vs. entity counts and optionally writes it to `outfile` and returns it.

table_latex (*fname*, *header*, *data*, *sum_row=True*, *row_order=None*, *latex_hdr=True*, *caption=''*, *font='HelveticaNeueLTStd-LtCn'*, *fontsize=8*, *sum_label='Total'*, *sum_cols=None*, *header_format='%s'*, *by_category=True*)

third_source_directions (*graph=None*, *use_string_effects=False*, *use_laudanna_data=False*)
This method calls a series of methods to get additional direction & effect information from sources having no literature curated references, but giving sufficient evidence about the directionality for interactions already supported by literature evidences from other sources.

tissue_network (*tissue*, *graph=None*)
Returns a network which includes the proteins expressed in certain tissue according to ProteomicsDB.

Parameters

- **tissue** (*str*) – Tissue name as used in ProteomicsDB.
- **graph** (*igraph.Graph*) – A graph object, by default the *graph* attribute of the current instance.

transcription_factors ()

uniprot (*uniprot*)
Returns `igraph.Vertex()` object if the UniProt can be found in the default undirected network, otherwise `None`.

@uniprot [str] UniProt ID.

uniprots(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default undirected graph.

uniq_node_list(*lst*)
Returns a given list of nodes containing only the unique elements.

Parameters **lst** (*list*) – List of nodes.

Returns (*list*) – Copy of *lst* containing only unique nodes.

uniq_ptm(*ptms*)

uniq_ptms()

up(*uniprot*)
Returns `igraph.Vertex()` object if the UniProt can be found in the default undirected network, otherwise `None`.

@uniprot [str] UniProt ID.

up_affected_by(*uniprot*)

up_affects(*uniprot*)

up_edge(*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

@source [str] UniProt ID

@target [str] UniProt ID

@directed [bool] To be passed to igraph.Graph.get_eid()

up_in_directed(*uniprot*)

up_in_undirected(*uniprot*)

up_inhibited_by(*uniprot*)

up_inhibits(*uniprot*)

up_neighborhood(*uniprots*, *order=1*, *mode='ALL'*)

up_neighbors(*uniprot*, *mode='ALL'*)

up_stimulated_by(*uniprot*)

up_stimulates(*uniprot*)

update_adjlist(*graph=None*, *mode='ALL'*)
Creates an adjacency list in a list of sets format.

update_attrs()
Updates the node and edge attributes. Note that no data is donwloaded, mainly updates the dictionaries of attributes `pypath.main.PyPath.edgeAttrs` and `pypath.main.PyPath.vertexAttrs` containing the attributes names and their correspoding types and initializes such attributes in the network nodes/edges if they weren't.

update_cats()
Makes sure that the `pypath.main.PyPath.has_cats` attribute is an up to date [set] of all categories in the current network.

update_db_dict()

update_pathway_types()
Updates the pathway types attribute (pypath.main.PyPath.pathway_types) according to the loaded resources of the undirected network.

update_pathways()
Makes sure that the pypath.main.PyPath.pathways attribute is an up to date [dict] of all pathways and their sources in the current network.

update_sources()
Makes sure that the pypath.main.PyPath.sources attribute is an up to date [list] of all sources in the current network.

update_summaries()
Creates a dict with many summarizing and comparative statistics about the resources in the current network. The result will be assigned to the attribute summaries.

update_vertex_sources()
Updates the all the vertex attributes 'sources' and 'references' according to their related edges (on the undirected graph).

update_vindex()
This is deprecated.

update_vname()
Fast lookup of node names and indexes, these are hold in a [list] and a [dict] as well. However, every time new nodes are added, these should be updated. This function is automatically called after all operations affecting node indices.

ups(*uniprots*)
Returns list of igraph.Vertex() object for a list of UniProt IDs omitting those could not be found in the default undirected graph.

v(*identifier*)
Returns igraph.Vertex() object if the identifier is a valid vertex index in the default undirected graph, or a UniProt ID or GeneSymbol which can be found in the default undirected network, otherwise None.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or igraph.Vertex object.

vertex_pathways()
Some resources assignes interactions some others proteins to pathways. This function copies pathway annotations from edge attributes to vertex attributes.

vsgs()
Returns a generator sequence of the node names as GeneSymbols [str] (from the undirected graph).

Returns (*generator*) – Sequence containing the node names as GeneSymbols [str].

vsup()
Returns a generator sequence of the node names as UniProt IDs [str] (from the undirected graph).

Returns (*generator*) – Sequence containing the node names as UniProt IDs [str].

wang_effects(*graph=None*)

write_table(*tbl*, *outfile*, *sep='\t'*, *cut=None*, *colnames=True*, *rownames=True*)
Writes a given table to a file.

Parameters

- **tbl** (*dict*) – Contains the data of the table. It is assumed that keys are the row names [str] and the values, well, values. Column names (if any) are defined with the key 'header'.

- **outfile** (*str*) – File name where to save the table. The file will be saved under the object's `pypath.main.PyPath.outdir` (`'results'` by default).
- **sep** (*str*) – Optional, `'	'` (tab) by default. Specifies the separator for the file.
- **cut** (*int*) – Optional, `None` by default. Specifies the maximum number of characters for the row names.
- **colnames** (*bool*) – Optional, `True` by default. Specifies whether to write the column names in the file or not.
- **rownames** (*bool*) – Optional, `True` by default. Specifies whether to write the row names in the file or not.

class pypath.legacy.main.**Direction**(*id_a*, *id_b*)

This is a *legacy* object for handling directionality information associated with unique pairs of interacting molecular entities. The py:class:*pypath.interaction.Interaction* available in the `attrs` edge attribute of the legacy `pypath.main.PyPath`` object provides a clearer and much more versatile interface. This object will be removed at some point, we don't recommend to build applications by using it.

Object storing directionality information of an edge. Also includes information about the reverse direction, mode of regulation and sources of that information.

Parameters

- **id_a** (*str*) – Name of the source node.
- **id_b** (*str*) – Name of the target node.

Variables

- **dirs** (*dict*) – Dictionary containing the presence of directionality of the given edge. Keys are *straight*, *reverse* and `'undirected'` and their values denote the presence/absence [bool].
- **negative** (*dict*) – Dictionary contianing the presence/absence [bool] of negative interactions for both `straight` and `reverse` directions.
- **negative_sources** (*dict*) – Contains the resource names [str] supporting a negative interaction on `straight` and `reverse` directions.
- **nodes** (*list*) – Contains the node names [str] sorted alphabetically (*id_a*, *id_b*).
- **positive** (*dict*) – Dictionary contianing the presence/absence [bool] of positive interactions for both `straight` and `reverse` directions.
- **positive_sources** (*dict*) – Contains the resource names [str] supporting a positive interaction on `straight` and `reverse` directions.
- **reverse** (*tuple*) – Contains the node names [str] in reverse order e.g. (*id_b*, *id_a*).
- ***sources*** (*dict*) – Contains the resource names [str] of a given edge for each directionality (`straight`, `reverse` and `'undirected'`). Values are sets containing the names of those resources supporting such directionality.
- **straight** (*tuple*) – Contains the node names [str] in the original order e.g. (*id_a*, *id_b*).

check_nodes(*nodes*)

Checks if *nodes* is contained in the edge.

Parameters **nodes** (*list*) – Or [tuple], contains the names of the nodes to be checked.

Returns (*bool*) – `True` if all elements in *nodes* are contained in the object `nodes` list.

check_param(*di*)
Checks if *di* is `'undirected'` or contains the nodes of the current edge. Used internally to check that *di* is a valid key for the object attributes declared on dictionaries.

Parameters **di** (*tuple*) – Or [str], key to be tested for validity.

Returns

(*bool*) – `True` if *di* is `'undirected'` or a tuple of node names contained in the edge, `False` otherwise.

consensus_edges()
Infers the consensus edge(s) according to the number of supporting sources. This includes direction and sign.

Returns (*list*) – Contains the consensus edge(s) along with the consensus sign. If there is no major directionality, both are returned. The structure is as follows: `['<source>', '<target>', '<(un)directed>', '<sign>']`

get_dir(*direction*, *sources=False*)
Returns the state (or *sources* if specified) of the given *direction*.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`). Pair of nodes from which direction information is to be retrieved.
- **sources** (*bool*) – Optional, `'False'` by default. Specifies if the `sources` information of the given direction is to be retrieved instead.

Returns (*bool* or *set*) – (if `sources=True`). Presence/absence of the requested direction (or the list of sources if specified). Returns `None` if *direction* is not valid.

get_direction(*direction*, *sources=False*)
Returns the state (or *sources* if specified) of the given *direction*.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`). Pair of nodes from which direction information is to be retrieved.
- **sources** (*bool*) – Optional, `'False'` by default. Specifies if the `sources` information of the given direction is to be retrieved instead.

Returns (*bool* or *set*) – (if `sources=True`). Presence/absence of the requested direction (or the list of sources if specified). Returns `None` if *direction* is not valid.

get_directions(*src*, *tgt*, *sources=False*)
Returns all directions with boolean values or list of sources.

Parameters

- **src** (*str*) – Source node.
- **tgt** (*str*) – Target node.
- **sources** (*bool*) – Optional, `False` by default. Specifies whether to return the `sources` attribute instead of `dirs`.

Returns Contains the `dirs` (or `sources` if specified) of the given edge.

get_dirs(*src*, *tgt*, *sources=False*)
Returns all directions with boolean values or list of sources.

Parameters

- **src** (*str*) – Source node.
- **tgt** (*str*) – Target node.
- **sources** (*bool*) – Optional, `False` by default. Specifies whether to return the `sources` attribute instead of `dirs`.

Returns Contains the `dirs` (or `sources` if specified) of the given edge.

get_sign(*direction*, *sign=None*, *sources=False*)
Retrieves the sign information of the edge in the given diretion. If specified in *sign*, only that sign's information will be retrieved. If specified in *sources*, the sources of that information will be retrieved instead.

Parameters

- **direction** (*tuple*) – Contains the pair of nodes specifying the directionality of the edge from which th information is to be retrieved.
- **sign** (*str*) – Optional, `None` by default. Denotes whether to retrieve the `'positive'` or `'negative'` specific information.
- **sources** (*bool*) – Optional, `False` by default. Specifies whether to return the sources instead of sign.

Returns (*list*) – If `sign=None` containing [bool] values denoting the presence of positive and negative sign on that direction, if `sources=True` the [set] of sources for each of them will be returned instead. If *sign* is specified, returns [bool] or [set] (if `sources=True`) of that specific direction and sign.

has_sign(*direction=None*, *resources=None*)
Checks whether the edge (or for a specific *direction*) has any signed information (about positive/negative interactions).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, only the information of that direction is checked for sign.

Returns

(*bool*) – `True` if there exist any information on the sign of the interaction, `False` otherwise.

is_directed()
Checks if edge has any directionality information.

Returns (*bool*) – Returns `True` if any of the `dirs` attribute values is `True` (except `'undirected'`), `False` otherwise.

is_directed_by_resources(*resources=None*)
Checks if edge has any directionality information from some resource(s).

Returns (*bool*) – Returns `True` if any of the `dirs` attribute values is `True` (except `'undirected'`), `False` otherwise.

is_inhibition(*direction=None*, *resources=None*)
Checks if any (or for a specific *direction*) interaction is inhibition (negative interaction).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, checks the `negative` attribute of that specific directionality. If not specified, checks both.

Returns (*bool*) – `True` if any interaction (or the specified *direction*) is inhibitory (negative).

is_mutual(*resources=None*)
Checks if the edge has mutual directions (both A–>B and B–>A).

is_mutual_by_resources(*resources=None*)
Checks if the edge has mutual directions (both A–>B and B–>A) according to some resource(s).

is_stimulation(*direction=None*, *resources=None*)
Checks if any (or for a specific *direction*) interaction is activation (positive interaction).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, checks the `positive` attribute of that specific directionality. If not specified, checks both.

Returns (*bool*) – `True` if any interaction (or the specified *direction*) is activatory (positive).

majority_dir()
Infers which is the major directionality of the edge by number of supporting sources.

Returns (*tuple*) – Contains the pair of nodes denoting the consensus directionality. If the number of sources on both directions is equal, `None` is returned. If there is no directionality information, `'undirected'` will be returned.

majority_sign()
Infers which is the major sign (activation/inhibition) of the edge by number of supporting sources on both directions.

Returns (*dict*) – Keys are the node tuples on both directions (`straight`/`reverse`) and values can be either `None` if that direction has no sign information or a list of two [bool] elements corresponding to majority of positive and majority of negative support. In case both elements of the list are `True`, this means the number of supporting sources for both signs in that direction is equal.

merge(*other*)
Merges current edge with another (if and only if they are the same class and contain the same nodes). Updates the attributes `dirs`, `sources`, `positive`, `negative`, `positive_sources` and `negative_sources`.

Parameters **other** (*pypath.main.Direction*) – The new edge object to be merged with the current one.

negative_reverse()
Checks if the `reverse` directionality is a negative interaction.

Returns (*bool*) – `True` if there is supporting information on the `reverse` direction of the edge as inhibition. `False` otherwise.

negative_sources_reverse()
Retrieves the list of sources for the `reverse` direction and negative sign.

Returns (*set*) – Contains the names of the sources supporting the `reverse` directionality of the edge with a negative sign.

negative_sources_straight()
Retrieves the list of sources for the `straight` direction and negative sign.

Returns (*set*) – Contains the names of the sources supporting the `straight` directionality of the edge with a negative sign.

negative_straight()
Checks if the `straight` directionality is a negative interaction.

Returns (*bool*) – `True` if there is supporting information on the `straight` direction of the edge as inhibition. `False` otherwise.

positive_reverse()
Checks if the `reverse` directionality is a positive interaction.

Returns (*bool*) – `True` if there is supporting information on the `reverse` direction of the edge as activation. `False` otherwise.

positive_sources_reverse()
Retrieves the list of sources for the `reverse` direction and positive sign.

Returns (*set*) – Contains the names of the sources supporting the `reverse` directionality of the edge with a positive sign.

positive_sources_straight()
Retrieves the list of sources for the `straight` direction and positive sign.

Returns (*set*) – Contains the names of the sources supporting the `straight` directionality of the edge with a positive sign.

positive_straight()
Checks if the `straight` directionality is a positive interaction.

Returns (*bool*) – `True` if there is supporting information on the `straight` direction of the edge as activation. `False` otherwise.

reload()
Reloads the object from the module level.

set_dir(*direction*, *source*)
Adds directionality information with the corresponding data source named. Modifies self attributes `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str], the directionality key for which the value on `dirs` has to be set `True`.
- **source** (*set*) – Contains the name(s) of the source(s) from which such information was obtained.

set_direction(*direction*, *source*)
Adds directionality information with the corresponding data source named. Modifies self attributes `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str], the directionality key for which the value on `dirs` has to be set `True`.
- **source** (*set*) – Contains the name(s) of the source(s) from which such information was obtained.

set_sign(*direction*, *sign*, *source*)
Sets sign and source information on a given direction of the edge. Modifies the attributes `positive` and `positive_sources` or `negative` and `negative_sources` depending on the sign. Direction is also updated accordingly, which also modifies the attributes `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Pair of edge nodes specifying the direction from which the information is to be set/updated.
- **sign** (*str*) – Specifies the type of interaction. If `'positive'`, is considered activation, otherwise, is assumed to be negative (inhibition).
- **source** (*set*) – Contains the name(s) of the source(s) from which the information was obtained.

source(*undirected=False*, *resources=None*)
Returns the name(s) of the source node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the source node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

sources_reverse()
Retrieves the list of sources for the `reverse` direction.

Returns (*set*) – Contains the names of the sources supporting the `reverse` directionality of the edge.

sources_straight()
Retrieves the list of sources for the `straight` direction.

Returns (*set*) – Contains the names of the sources supporting the `straight` directionality of the edge.

sources_undirected()
Retrieves the list of sources without directed information.

Returns (*set*) – Contains the names of the sources supporting the edge presence but without specific directionality information.

src(*undirected=False*, *resources=None*)
Returns the name(s) of the source node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the source node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

src_by_source(*source*)
Returns the name(s) of the source node(s) for each existing direction on the interaction for a specific *source*.

Parameters **source** (*str*) – Name of the source according to which the information is to be retrieved.

Returns (*list*) – Contains the name(s) for the source node(s) according to the specified *source*. This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the specified *source* is not found or invalid, an empty list will be returned.

target(*undirected=False*, *resources=None*)
Returns the name(s) of the target node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the target node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

tgt(*undirected=False*, *resources=None*)
Returns the name(s) of the target node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the target node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

tgt_by_source(*source*)
Returns the name(s) of the target node(s) for each existing direction on the interaction for a specific *source*.

Parameters **source** (*str*) – Name of the source according to which the information is to be retrieved.

Returns (*list*) – Contains the name(s) for the target node(s) according to the specified *source*. This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the specified *source* is not found or invalid, an empty list will be returned.

translate(*ids*)
Translates the node names/identifiers according to the dictionary *ids*.

Parameters **ids** (*dict*) – Dictionary containing (at least) the current names of the nodes as keys and their translation as values.

Returns (*pypath.main.Direction*) – The copy of current edge object with translated node names.

unset_dir(*direction*, *source=None*)
Removes directionality and/or source information of the specified *direction*. Modifies attribute `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`) the pair of nodes specifying the directionality from which the information is to be removed.
- **source** (*set*) – Optional, `None` by default. If specified, determines which specific source(s) is(are) to be removed from `sources` attribute in the specified *direction*.

unset_direction(*direction*, *source=None*)
Removes directionality and/or source information of the specified *direction*. Modifies attribute `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`) the pair of nodes specifying the directionality from which the information is to be removed.
- **source** (*set*) – Optional, `None` by default. If specified, determines which specific source(s) is(are) to be removed from `sources` attribute in the specified *direction*.

unset_sign(*direction*, *sign*, *source=None*)
Removes sign and/or source information of the specified *direction* and *sign*. Modifies attribute `positive` and `positive_sources` or `negative` and `negative_sources` (or `positive_attributes`/`negative_sources` only if `source=True`).

Parameters

- **direction** (*tuple*) – The pair of nodes specifying the directionality from which the information is to be removed.
- **sign** (*str*) – Sign from which the information is to be removed. Must be either `'positive'` or `'negative'`.
- **source** (*set*) – Optional, `None` by default. If specified, determines which source(s) is(are) to be removed from the sources in the specified *direction* and *sign*.

which_directions(*resources=None*, *effect=None*)
Returns the pair(s) of nodes for which there is information about their directionality.

Parameters

- **effect** (*str*) – Either *positive* or *negative*.

- **resources** (*str, set*) – Limits the query to one or more resources. Optional.

Returns (*tuple*) – Tuple of tuples with pairs of nodes where the first element is the source and the second is the target entity, according to the given resources and limited to the effect.

which_dirs(*resources=None*, *effect=None*)
Returns the pair(s) of nodes for which there is information about their directionality.

Parameters

- **effect** (*str*) – Either *positive* or *negative*.
- **resources** (*str, set*) – Limits the query to one or more resources. Optional.

Returns (*tuple*) – Tuple of tuples with pairs of nodes where the first element is the source and the second is the target entity, according to the given resources and limited to the effect.

which_signs(*resources=None*, *effect=None*)
Returns the pair(s) of nodes for which there is information about their effect signs.

Parameters

- **resources** (*str, set*) – Limits the query to one or more resources. Optional.
- **effect** (*str*) – Either *positive* or *negative*, limiting the query to positive or negative effects; for any other values effects of both signs will be returned.

Returns (*tuple*) – Tuple of tuples with pairs of nodes where the first element is a tuple of the source and the target entity, while the second element is the effect sign, according to the given resources. E.g. ((('A', 'B'), 'positive'),)

2.24 mirbase

pypath.inputs.mirbase.**get_mirbase_aliases**(*organism=9606*)
Downloads and processes mapping tables from miRBase.

pypath.inputs.mirbase.**get_uniprot_sec**(*organism=9606*)
Downloads and processes the mapping between secondary and primary UniProt IDs.

Yields pairs of secondary and primary UniProt IDs.

Parameters **organism** (*int*) – NCBI Taxonomy ID of the organism.

2.25 mapping

class pypath.utils.mapping.**MapReader**(*param*, *ncbi_tax_id=None*, *entity_type=None*, *load_a_to_b=True*, *load_b_to_a=False*, *uniprots=None*, *lifetime=300*)
Reads ID translation data and creates `MappingTable` instances. When initializing ID conversion tables for the first time data is downloaded from UniProt and read into dictionaries. It takes a couple of seconds. Data is saved to pickle dumps, this way later the tables load much faster.

Parameters **str** (*source_type*) – Type of the resource, either *file*, *uniprot* or *unprotlist*.

load()
The complete process of loading mapping tables. First sets up the paths of the cache files, then loads the tables from the cache files or the original sources if necessary. Upon successful loading from an original source writes the results to cache files.

mapping_table_a_to_b
Returns a `MappingTable` instance created from the already loaded data.

mapping_table_b_to_a
Returns a `MappingTable` instance created from the already loaded data.

read()
Reads the ID translation data from the original source.

read_cache()
Reads the ID translation data from a previously saved pickle file.

read_mapping_uniprot()
Downloads ID mappings directly from UniProt. See the names of possible identifiers here: http://www.uniprot.org/help/programmatic_access

read_mapping_uniprot_list()
Builds a mapping table by downloading data from UniProt's upload lists service.

set_uniprot_space(*swissprot=None*)
Sets up a search space of UniProt IDs.

setup_cache()
Constructs the cache file path as md5 hash of the parameters.

tables_loaded()
Tells if the requested tables have been created.

write_cache()
Exports the ID translation data into pickle files.

class pypath.utils.mapping.**MappingTable**(*data*, *id_type*, *target_id_type*, *ncbi_tax_id*, *lifetime=300*)
This is the class directly handling ID translation data. It does not care about loading it or what kind of IDs these only accepts the translation dictionary.

lifetime [int] If this table has not been used for longer than this preiod it is to be removed at next cleanup. Time in seconds.

2.26 maps

2.27 network

class pypath.core.network.**Network**(*resources=None*, *make_df=False*, *df_by_source=False*, *df_with_references=False*, *df_columns=None*, *df_dtype=None*, *pickle_file=None*, *ncbi_tax_id=9606*, *allow_loops=True*, ***kwargs*)
Represents a molecular interaction network. Provides various methods to query the network and its components. Optionally converts the network to a `pandas.DataFrame` of interactions.

Parameters

- **resources** (*list, dict*) – One or more lists or dictionaries containing `pypath.resource.NetworkResource` objects.
- **make_df** (*bool*) – Create a `pandas.DataFrame` already when creating the instance. If no network data loaded no data frame will be created.

- **ncbi_tax_id** (*int*) – Restrict the network only to this organism. If `None` identifiers from any organism will be allowed.
- **allow_loops** (*bool*) – Allow interactions with the their two endpoints being the same entity.

activated_by()

Parameters

- **entity** (*str*, `Entity`, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

activates()

Parameters

- **entity** (*str*, `Entity`, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

add_interaction(*interaction*, *attrs=None*, *only_directions=False*)

Adds a ready `pypath.interaction.Interaction` object to the network. If an interaction between the two endpoints already exists, the interactions will be merged: this stands for the directions, signs, evidences and other attributes.

Parameters

- **interaction** (`interaction.Interaction`) – A `pypath.interaction.Interaction` object.
- **attrs** (*NoneType*, *dict*) – Optional, a dictionary of extra (usually resource specific) attributes.
- **only_directions** (*bool*) – If the interaction between the two endpoints does not exist it won't be added to the network. Otherwise all attributes (direction, effect sign, evidences, etc) will be merged to the existing interaction. Apart from the endpoints also the `interaction_type` of the existing interaction has to match the interaction added here.

add_node(*entity*, *attrs=None*, *add=True*)

Adds a molecular entity to the py:attr:`nodes` and py:attr:`nodes_by_label` dictionaries.

Parameters

- **entity** (`entity.Entity`) – An object representing a molecular entity.
- **attrs** (*NoneType*, *dict*) – Optional extra attributes to be assigned to the entity.

- **add** (*bool*) – Whether to add a new molecular entity to the network if it does not exist yet. If `False` will only update attributes for existing entities otherwise will do nothing.

collect_complex_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *complex_identifiers* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_complex_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *complex_labels* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_complexes(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *complexes* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_curation_effort(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *curation_effort* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_data_models(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *data_models* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_directed(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_directed* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_directed_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_directed_in* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_directed_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_directed_out* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_negative(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_negative* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_negative_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_negative_in* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_negative_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_negative_out* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_non_directed(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_non_directed* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_positive(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_positive* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_positive_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_positive_in* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_positive_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_positive_out* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_signed(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_signed* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_signed_in(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_signed_in* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_signed_out(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_signed_out* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_degrees_undirected(*direction=None*, *effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *degrees_undirected* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_entities(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *entities* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_evidences(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *evidences* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *identifiers* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interaction_types(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interaction_types* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_0(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_0* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_directed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_directed* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_mutual(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_mutual* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_negative(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_negative* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_non_directed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_non_directed* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_non_directed_0(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_non_directed_0* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_positive(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_positive* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_signed(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_signed* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_undirected(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_undirected* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_interactions_undirected_0(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *interactions_undirected_0* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *labels* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_lncrna_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *lncrna_identifiers* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_lncrna_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *lncrna_labels* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_lncrnas(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *lncrnas* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_mirna_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *mirna_identifiers* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_mirna_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *mirna_labels* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_mirnas(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *mirnas* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_protein_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *protein_identifiers* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_protein_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *protein_labels* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_proteins(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *proteins* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_references(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *references* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_resource_names(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *resource_names* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_resource_names_via(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)

Builds a comprehensive collection of *resource_names_via* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_resources(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *resources* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_resources_via(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *resources_via* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_small_molecule_identifiers(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *small_molecule_identifiers* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_small_molecule_labels(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *small_molecule_labels* entities across the network, counts unique and shared objects by resource, data model and interaction types.

collect_small_molecules(*effect=None*, *resources=None*, *data_model=None*, *interaction_type=None*, *via=None*, *references=None*)
Builds a comprehensive collection of *small_molecules* entities across the network, counts unique and shared objects by resource, data model and interaction types.

complex_identifiers_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_identifiers_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_identifiers_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_identifiers_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_identifiers_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_identifiers_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_labels_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_labels_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_labels_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_labels_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_labels_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complex_labels_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complexes_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complexes_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complexes_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complexes_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complexes_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

complexes_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

count_activated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_activates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_complex_identifiers()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_identifiers_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_identifiers_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_identifiers_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_identifiers_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_identifiers_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_identifiers_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complex_labels_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_complexes_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_curation_effort_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_data_models_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_in_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_directed_out_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_in_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_negative_out_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_non_directed_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_in_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_positive_out_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_in_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_signed_out_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_degrees_undirected_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_entities_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_evidences_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_identifiers_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interaction_types_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_0_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_directed_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_mutual_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_negative_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_0_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_non_directed_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_positive_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_signed_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_0_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_interactions_undirected_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_labels_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_identifiers_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrna_labels_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_lncrnas_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_identifiers_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirna_labels_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_mirnas_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_partners(*entity*, ***kwargs*)
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_transcriptionally_activated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_transcriptionally_activates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_transcriptionally_regulated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_transcriptionally_regulates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_transcriptionally_suppressed_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_transcriptionally_suppresses()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_translationally_activated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_translationally_activates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_translationally_regulated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_translationally_regulates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_translationally_suppressed_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_post_translationally_suppresses()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_protein_identifiers()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_identifiers_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_identifiers_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_identifiers_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_identifiers_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_identifiers_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_identifiers_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_protein_labels_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_proteins_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_references_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_regulated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_regulates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_resource_names()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resource_names_via_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_resources_via_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns

the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_identifiers_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecule_labels_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules_by_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules_by_interaction_type()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules_by_interaction_type_and_data_model()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules_by_interaction_type_and_data_model_and_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules_by_reference()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_small_molecules_by_resource()
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

count_suppressed_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_suppresses()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_transcriptionally_activated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_transcriptionally_activates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_transcriptionally_regulated_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_transcriptionally_regulates()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_transcriptionally_suppressed_by()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

count_transcriptionally_suppresses()
Returns the count of the interacting partners for one or more entities according to the specified criteria. Please refer to the docs of the `partners` method.

curation_effort_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

curation_effort_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

curation_effort_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

curation_effort_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

curation_effort_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

curation_effort_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

data_models_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

data_models_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

data_models_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

data_models_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

data_models_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

data_models_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_in_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_in_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_in_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_in_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_in_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_in_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_out_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_out_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_out_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_out_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_out_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_directed_out_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_in_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_in_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_in_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_in_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_in_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_in_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_out_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_out_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_out_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_out_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_out_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_negative_out_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_non_directed_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_non_directed_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_non_directed_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_non_directed_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_non_directed_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_non_directed_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_in_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_in_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_in_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_in_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_in_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_in_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_out_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_out_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_out_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_out_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_out_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_positive_out_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_in_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_in_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_in_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_in_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_in_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_in_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_out_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_out_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_out_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_out_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_out_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_signed_out_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_undirected_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_undirected_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_undirected_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_undirected_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_undirected_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

degrees_undirected_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

classmethod dorothea(*levels=None*, *ncbi_tax_id=9606*, ***kwargs*)
Initializes a new `Network` object with loading the transcriptional regulation network from DoRothEA.

Parameters **levels** (*NontType, set*) – The confidence levels to include.

entities_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

entities_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

entities_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

entities_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

entities_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

entities_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

entity_by_id(*identifier*)
Returns a `pypath.entity.Entity` object representing a molecular entity by looking it up by its identifier. If the molecule does not present in the current network `None` will be returned.

Parameters **identifier** (*str*) – The identifier of a molecular entity. Unless it's been set otherwise for genes/proteins it is the UniProt ID. E.g. `'P00533'`.

entity_by_label(*label*)
Returns a `pypath.entity.Entity` object representing a molecular entity by looking it up by its label. If the molecule does not present in the current network `None` will be returned.

Parameters **label** (*str*) – The label of a molecular entity. Unless it's been set otherwise for genes/proteins it is the Gene Symbol. E.g. `'EGFR'`.

evidences_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

evidences_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

evidences_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

evidences_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

evidences_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

evidences_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

extra_directions(*resources='extra_directions'*, *use_laudanna=False*, *use_string=False*)
Adds additional direction & effect information from resources having no literature curated references, but giving sufficient evidence about the directionality for interactions already supported by literature evidences from other sources.

find_paths(*start*, *end=None*, *loops=False*, *mode='OUT'*, *maxlen=2*, *minlen=1*, *direction=None*, *effect=None*, *resources=None*, *interaction_type=None*, *data_model=None*, *via=None*, *references=None*, *silent=False*)
Finds all paths up to length `maxlen` between groups of nodes. In addition is able to search for motifs or select the nodes of a subnetwork around certain nodes.

Parameters

- **start** (*str*, Entity, *list*, *tuple*, *set*, *EntityList*) – Starting node(s) of the paths.
- **end** (*str*, Entity, *list*, *tuple*, *set*, *EntityList*, *NoneType*) – Target node(s) of the paths. If `None` any target node will be accepted and all paths from the starting nodes with length `maxlen` will be returned.
- **loops** (*bool*) – Search for loops, i.e. the start and end nodes of each path should be the same.
- **mode** (*str*) – Direction of the paths. `'OUT'` means from `start` to `end`, `'IN'` the opposite direction while `'ALL'` both directions.
- **maxlen** (*int*) – Maximum length of paths in steps, i.e. if maxlen = 3, then the longest path may consist of 3 edges and 4 nodes.
- **minlen** (*int*) – Minimum length of the path.
- **silent** (*bool*) – Indicate progress by showing a progress bar.

Details

The arguments: `direction`, `effect`, `resources`, `interaction_type`, `data_model`, `via` and `references` will be passed to the `partners` method of this object and from there to the relevant methods of the `Interaction` and `Evidence` objects. By these arguments it is possible to filter the interactions in the paths according to custom criteria. If any of these arguments is a `tuple` or `list`, its first value will be used to match the first interaction in the path, the second for the second one and so on. If the list or tuple is shorter then `maxlen`, its last element will be used for all interactions. If it's longer than `maxlen`, the remaining elements will be discarded. This way the method is able to search for custom motives. For example, let's say you want to find the motives where the estrogen receptor transcription factor *ESR1* transcriptionally regulates a gene encoding a protein which then has some effect post-translationally on *ESR1*:

```
>>> n.find_paths(
...     'ESR1',
...     loops = True,
...     minlen = 2,
...     interaction_type = ('transcriptional', 'post_translational'),
... )
```

Or if you are interested only in the -/+ feedback loops i.e. *ESR1 –(-)–> X –(+)–> ESR1*:

```
>>> n.find_paths(
...     'ESR1',
...     loops = True,
...     minlen = 2,
...     interaction_type = ('transcriptional', 'post_translational'),
...     effect = ('negative', 'positive'),
... )
```

classmethod from_igraph(*pa*, ***kwargs*)

Creates an instance from an `igraph.Graph` based `pypath.main.PyPath` object.

Parameters **pa** (*pypath.main.PyPath*) – A `pypath.main.PyPath` object with network data loaded.

classmethod from_pickle(*pickle_file*, ***kwargs*)

Initializes a new `Network` object by loading it from a pickle file. Returns a `Network` object.

Parameters **pickle_file** (*str*) – Path to a pickle file.

****kwargs:** Passed to `Network.__init__`.

get_complex_identifiers()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_complex_labels()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_complexes()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_curation_effort()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_data_models()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_directed()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_directed_in()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_directed_out()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_negative()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_negative_in()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_negative_out()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_non_directed()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_positive()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_positive_in()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_positive_out()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_signed()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_signed_in()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_signed_out()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_degrees_undirected()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_entities()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_evidences()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_identifiers()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interaction_types()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_0()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_directed()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_mutual()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_negative()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_non_directed()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_non_directed_0()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_positive()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_signed()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_undirected()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_interactions_undirected_0()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_labels()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_lncrna_identifiers()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_lncrna_labels()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_lncrnas()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_mirna_identifiers()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_mirna_labels()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_mirnas()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_organisms()
Returns the set of all NCBI Taxonomy IDs occurring in the network.

get_protein_identifiers()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_protein_labels()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_proteins()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_references()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_resource_names()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_resource_names_via()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_resources()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_resources_via()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_small_molecule_identifiers()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_small_molecule_labels()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

get_small_molecules()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

identifiers_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

identifiers_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

identifiers_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

identifiers_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

identifiers_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

identifiers_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

init_network(*resources=None*, *make_df=False*, *exclude=None*, *reread=False*, *redownload=False*, *keep_raw=False*, *top_call=True*, *cache_files=None*, *only_directions=False*, *pickle_file=None*)

Loads data from a network resource or a collection of resources.

Parameters

- **resources** (*str,dict,list,resource.NetworkResource*) – An object defining one or more network resources. If *str* it will be looked up among the collections in the `pypath.resources.network` module (e.g. `'pathway'` will load all resources in the *pathway* collection). If *dict* or *list* it will be processed recursively i.e. the `load` method will be called for each element. If it is a `pypath.resource.NetworkResource` object it will be processed and added to the network.
- **make_df** (*bool*) – Whether to create a `pandas.DataFrame` after loading all resources.
- **exclude** (*NoneType,set*) – A *set* of resource names to be ignored. It is useful if you want to load a collection with the exception of a few resources.

interaction(*a*, *b*)

Retrieves the interaction *a* –> *b* if it exists in the network, otherwise *b* –> *a*. If no interaction exist between *a* and *b* returns *None*.

interaction_by_id(*id_a*, *id_b*)

Returns a `pypath.interaction.Interaction` object by looking it up based on a pair of identifiers. If the interaction does not exist in the network `None` will be returned.

Parameters

- **id_a** (*str*) – The identifier of one of the partners in the interaction. Unless it's been set otherwise for genes/proteins it is the UniProt ID. E.g. `'P00533'`.
- **id_b** (*str*) – The other partner, similarly to `id_a`. The order of the partners does not matter here.

interaction_by_label(*label_a*, *label_b*)

Returns a `pypath.interaction.Interaction` object by looking it up based on a pair of labels. If the interaction does not exist in the network `None` will be returned.

Parameters

- **label_a** (*str*) – The label of one of the partners in the interaction. Unless it's been set otherwise for genes/proteins it is the Gene Symbol. E.g. `'EGFR'`.
- **label_b** (*str*) – The other partner, similarly to `label_a`. The order of the partners does not matter here.

interaction_types_by_data_model()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interaction_types_by_interaction_type()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interaction_types_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interaction_types_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interaction_types_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interaction_types_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_0_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_0_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_0_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_0_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_0_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_0_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_directed_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_directed_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_directed_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_directed_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_directed_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_directed_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_mutual_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_mutual_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_mutual_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_mutual_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_mutual_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_mutual_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_negative_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_negative_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_negative_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_negative_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_negative_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_negative_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_0_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_0_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_0_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_0_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_0_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_0_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_non_directed_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_positive_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_positive_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_positive_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_positive_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_positive_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_positive_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_signed_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_signed_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_signed_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_signed_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_signed_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_signed_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_0_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_0_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_0_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_0_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_0_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_0_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

interactions_undirected_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

labels_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

labels_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

labels_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

labels_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

labels_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

labels_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_identifiers_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_identifiers_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_identifiers_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_identifiers_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_identifiers_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_identifiers_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_labels_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_labels_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_labels_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_labels_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_labels_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrna_labels_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrnas_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrnas_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrnas_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrnas_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrnas_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

lncrnas_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

load(*resources=None*, *make_df=False*, *exclude=None*, *reread=False*, *redownload=False*, *keep_raw=False*, *top_call=True*, *cache_files=None*, *only_directions=False*, *pickle_file=None*)
Loads data from a network resource or a collection of resources.

Parameters

- **resources** (*str,dict,list,resource.NetworkResource*) – An object defining one or more network resources. If *str* it will be looked up among the collections in the `pypath.resources.network` module (e.g. `'pathway'` will load all resources in the *pathway* collection). If *dict* or *list* it will be processed recursively i.e. the `load` method will be called for each element. If it is a `pypath.resource.NetworkResource` object it will be processed and added to the network.
- **make_df** (*bool*) – Whether to create a `pandas.DataFrame` after loading all resources.

- **exclude** (*NoneType, set*) – A *set* of resource names to be ignored. It is useful if you want to load a collection with the exception of a few resources.

load_from_pickle(*pickle_file*)
Loads the network to a pickle file.

Parameters **pickle_file** (*str*) – Path to the pickle file.

load_resource(*resource*, *clean=True*, *reread=None*, *redownload=None*, *keep_raw=False*, *only_directions=False*, ***kwargs*)
Loads the data from a single resource and attaches it to the network

Parameters

- **resource** (*pypath.input_formats.NetworkInput*) – `pypath.input_formats.NetworkInput` instance containing the detailed definition of the input format to the downloaded file.
- **clean** (*bool*) – Legacy parameter, has no effect at the moment. Optional, `True` by default. Whether to clean the graph after importing the data or not. See `pypath.main.PyPath.clean_graph()` for more information.
- **cache_files** (*dict*) – Legacy parameter, has no effect at the moment. Optional, `{}` by default. Contains the resource name(s) [str] (keys) and the corresponding cached file name [str]. If provided (and file exists) bypasses the download of the data for that resource and uses the cache file instead.
- **reread** (*bool*) – Optional, `False` by default. Specifies whether to reread the data files from the cache or omit them (similar to *redownload*).
- **redownload** (*bool*) – Optional, `False` by default. Specifies whether to re-download the data and ignore the cache.
- **only_directions** (*bool*) – If `True`, no new interactions will be created but direction and effect sign evidences will be added to existing interactions.

load_resources(*resources=None*, *make_df=False*, *exclude=None*, *reread=False*, *redownload=False*, *keep_raw=False*, *top_call=True*, *cache_files=None*, *only_directions=False*, *pickle_file=None*)
Loads data from a network resource or a collection of resources.

Parameters

- **resources** (*str,dict,list,resource.NetworkResource*) – An object defining one or more network resources. If *str* it will be looked up among the collections in the `pypath.resources.network` module (e.g. `'pathway'` will load all resources in the *pathway* collection). If *dict* or *list* it will be processed recursively i.e. the `load` method will be called for each element. If it is a `pypath.resource.NetworkResource` object it will be processed and added to the network.
- **make_df** (*bool*) – Whether to create a `pandas.DataFrame` after loading all resources.
- **exclude** (*NoneType, set*) – A *set* of resource names to be ignored. It is useful if you want to load a collection with the exception of a few resources.

make_df(*records=None*, *by_source=None*, *with_references=None*, *columns=None*, *dtype=None*)
Creates a `pandas.DataFrame` from the interactions.

mirna_identifiers_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_identifiers_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_identifiers_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_identifiers_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_identifiers_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_identifiers_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_labels_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_labels_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_labels_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_labels_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_labels_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirna_labels_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

classmethod mirna_target(*resources=None*, *make_df=None*, *reread=False*, *redownload=False*, *exclude=None*, *ncbi_tax_id=9606*, ***kwargs*)
Initializes a new `Network` object with loading a miRNA-mRNA regulation network from all databases by default.

**kwargs: passed to `Network.__init__`.

mirnas_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirnas_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirnas_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirnas_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirnas_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

mirnas_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

numof_interactions_per_reference()
Counts the number of interactions for each literature reference. Returns a `collections.Counter` object (similar to `dict`).

organisms_check(*organisms=None*, *remove_mismatches=True*, *remove_nonspecific=False*)
Scans the network for one or more organisms and removes the nodes and interactions which belong to any other organism.

Parameters

- **organisms** (*int*, *set*, *NoneType*) – One or more NCBI Taxonomy IDs. If `None` the value in `ncbi_tax_id` will be used. If that's too is `None` then only the entities with discrepancy between their stated organism and their identifier.
- **remove_mismatches** (*bool*) – Remove the entities where their `identifier` can not be found in the reference list from the database for their `taxon`.
- **remove_nonspecific** (*bool*) – Remove the entities with taxonomy ID zero, which is used to represent the non taxon specific entities such as metabolites or drug compounds.

partners(*entity*, *mode='ALL'*, *direction=None*, *effect=None*, *resources=None*, *interaction_type=None*, *data_model=None*, *via=None*, *references=None*, *return_interactions=False*)

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

post_transcriptionally_activated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

post_transcriptionally_activates()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

post_transcriptionally_regulated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

post_transcriptionally_regulates()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

post_transcriptionally_suppressed_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

post_transcriptionally_suppresses()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

post_translationally_activated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

post_translationally_activates()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

post_translationally_regulated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

post_translationally_regulates()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

post_translationally_suppressed_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

post_translationally_suppresses()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

protein_identifiers_by_data_model()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_identifiers_by_interaction_type()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_identifiers_by_interaction_type_and_data_model()

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_identifiers_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_identifiers_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_identifiers_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_labels_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_labels_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_labels_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_labels_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_labels_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

protein_labels_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

proteins_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

proteins_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

proteins_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

proteins_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

proteins_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

proteins_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

references_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

references_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

references_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

references_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

references_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

references_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

regulated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

regulates()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

reload()
Reloads the object from the module level.

remove_interaction(*entity_a*, *entity_b*)
Removes the interaction between two nodes if exists.

Parameters **entity_a,entity_b** (*str,* `Entity`) – A pair of molecular entity identifiers, labels or `Entity` objects.

remove_loops()
Removes the loop interactions from the network i.e. the ones with their two endpoints being the same entity.

remove_node(*entity*)
Removes a node with all its interactions. If the removal of the interactions leaves any of the partner nodes without interactions it will be removed too.

Parameters **entity** (*str,* `Entity`) – A molecular entity identifier, label or `Entity` object.

remove_zero_degree()
Removes all nodes with no interaction.

reset()
Removes network data i.e. creates empty interaction and node dictionaries.

resource_names
Returns a set of all resource names.

resource_names_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_via_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_via_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_via_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_via_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_via_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resource_names_via_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources
Returns a set of all resources.

resources_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_via_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_via_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_via_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_via_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_via_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

resources_via_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

save_to_pickle(*pickle_file*)
Saves the network to a pickle file.

Parameters **pickle_file** (*str*) – Path to the pickle file.

small_molecule_identifiers_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_identifiers_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_identifiers_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_identifiers_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_identifiers_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_identifiers_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_labels_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_labels_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_labels_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_labels_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_labels_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecule_labels_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecules_by_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecules_by_interaction_type()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecules_by_interaction_type_and_data_model()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecules_by_interaction_type_and_data_model_and_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecules_by_reference()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

small_molecules_by_resource()
Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

summaries_tab(*outfile=None*, *return_table=False*, *label_type=1*)
Creates a table from resource vs. entity counts and optionally writes it to `outfile` and returns it.

suppressed_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

suppresses()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

to_igraph()

Converts the network to the legacy igraph.Graph based PyPath object.

classmethod transcription(*dorothea=True*, *original_resources=True*, *dorothea_levels=None*, *exclude=None*, *reread=False*, *redownload=False*, *make_df=False*, *ncbi_tax_id=9606*, ***kwargs*)

Initializes a new Network object with loading a transcriptional regulation network from all databases by default.

**kwargs: passed to Network.__init__.

transcriptionally_activated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

transcriptionally_activates()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns EntityList object containing the partners having interactions to the queried node(s) matching all the criteria. If entity doesn't present in the network the returned EntityList will be empty just like if no interaction matches the criteria.

transcriptionally_regulated_by()

Parameters

- **entity** (*str*, Entity, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an Entity object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.

- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

transcriptionally_regulates()

Parameters

- **entity** (*str*, `Entity`, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

transcriptionally_suppressed_by()

Parameters

- **entity** (*str*, `Entity`, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

transcriptionally_suppresses()

Parameters

- **entity** (*str*, `Entity`, *list*, *set*, *tuple*, *EntityList*) – An identifier or label of a molecular entity or an `Entity` object. Alternatively an iterator with the elements of any of the types valid for a single entity argument, e.g. a list of gene symbols.
- **mode** (*str*) – Mode of counting the interactions: *IN*, *OUT* or *ALL* , whether to consider incoming, outgoing or all edges, respectively, respective to the *node defined in 'entity'*.

Returns `EntityList` object containing the partners having interactions to the queried node(s) matching all the criteria. If `entity` doesn't present in the network the returned `EntityList` will be empty just like if no interaction matches the criteria.

class pypath.core.network.**NetworkStatsRecord**(*total*, *by_resource*, *by_category*, *shared*, *unique*, *percent*, *shared_res_cat*, *unique_res_cat*, *percent_res_cat*, *shared_cat*, *unique_cat*, *percent_cat*, *resource_cat*, *cat_resource*, *method*, *label*)

by_category
Alias for field number 2

by_resource
Alias for field number 1

cat_resource
Alias for field number 13

label
Alias for field number 15

method
Alias for field number 14

percent
Alias for field number 5

percent_cat
Alias for field number 11

percent_res_cat
Alias for field number 8

resource_cat
Alias for field number 12

shared
Alias for field number 3

shared_cat
Alias for field number 9

shared_res_cat
Alias for field number 6

total
Alias for field number 0

unique
Alias for field number 4

unique_cat
Alias for field number 10

unique_res_cat
Alias for field number 7

2.28 omnipath

2.29 pdb

class pypath.utils.pdb.**ResidueMapper**
This class stores and serves the PDB –> UniProt residue level mapping. Attempts to download the mapping, and stores it for further use. Converts PDB residue numbers to the corresponding UniProt ones.

clean()
Removes cached mappings, freeing up memory.

2.30 plot

pypath.visual.plot.**is_opentype_cff_font**(*filename*)
This is necessary to fix a bug in matplotlib: https://github.com/matplotlib/matplotlib/pull/6714 Returns True

if the given font is a Postscript Compact Font Format Font embedded in an OpenType wrapper. Used by the PostScript and PDF backends that can not subset these fonts.

pypath.visual.plot.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, *randn* generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use *numpy.random.standard_normal* instead.

d0, d1, ..., dn [int, optional] The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Z [ndarray or float] A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

standard_normal : Similar, but takes a tuple as its argument.

For random samples from $N(\mu, \sigma^2)$, use:

`sigma * np.random.randn(...) + mu`

```
>>> np.random.randn()
2.1923875335537315 #random
```

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],  #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

2.31 progress

class pypath.share.progress.**Progress**(*total=None*, *name='Progress'*, *interval=None*, *percent=True*, *status='initializing'*, *done=0*, *init=True*, *unit='it'*, *off=None*)

Before I had my custom progressbar here. Now it is a wrapper around the great progressbar *tqdm*. Old implementation moved to *OldProgress* class.

get_desc()

Returns a formatted string of the description, consisted of the name and the status. The name supposed something constant within the life of the progressbar, while the status is there to give information about the current stage of the task.

init_tqdm()

Creates a tqdm instance.

set_done(*done*)

Sets the position of the progress bar.

set_status(*status*)

Changes the prefix of the progressbar.

set_total(*total*)
Changes the total value of the progress bar.

step(*step=1*, *msg=None*, *status='busy'*, *force=False*)
Updates the progressbar by the desired number of steps.

Parameters **step** (*int*) – Number of steps or items.

terminate(*status='finished'*)
Terminates the progressbar and destroys the tqdm object.

2.32 enz_sub

2.33 pyreact

class pypath.utils.pyreact.**BioPaxReader**(*biopax*, *source*, *cleanup_period=800*, *file_from_archive=None*, *silent=False*)
This class parses a BioPAX file and exposes its content easily accessible for further processing. First it opens the file, if necessary it extracts from the archive. Then an *lxml.etree.iterparse* object is created, so the iteration is efficient and memory requirements are minimal. The iterparse object is iterated then, and for each tag included in the *BioPaxReader.methods* dict, the appropriate method is called. These me- thods extract information from the BioPAX entity, and store it in arbit- rary data structures: strings, lists or dicts. These are stored in dicts where keys are the original IDs of the tags, prefixed with the unique ID of the parser object. This is necessary to give a way to merge later the result of parsing more BioPAX files. For example, *id42* may identify EGFR in one file, but AKT1 in the other. Then, the parser of the first file has a unique ID of a 5 letter random string, the second parser a different one, and the molecules with the same ID can be distinguished at merging, e.g. EGFR will be *ffjh2@id42* and AKT1 will be *tr9gy@id42*. The methods and the resulted dicts are named after the BioPAX elements, sometimes abbreviated. For example, *BioPaxReader.protein()* processes the <*bp:Protein*> elements, and stores the results in *BioPaxReader.proteins*.

In its current state, this class does not parse every information and all BioPax entities. For example, nucleic acid related entities and interactions are omitted. But these easily can be added with minor mo- difications.

biopax_size()
Gets the uncompressed size of the BioPax XML. This is needed in order to have a progress bar. This method should not be called directly, `BioPaxReader.process()` calls it.

cleanup_hook()
Removes the used elements to free up memory. This method should not be called directly, `BioPaxReader.iterate()` calls it.

close_biopax()
Deletes the iterator and closes the file object. This method should not be called directly, `BioPaxReader.process()` calls it.

extract()
Extracts the BioPax file from compressed archive. Creates a temporary file. This is needed to trace the progress of processing, which is useful in case of large files. This method should not be called directly, `BioPaxReader.process()` calls it.

init_etree()
Creates the `lxml.etree.iterparse` object. This method should not be called directly, `BioPaxReader.process()` calls it.

iterate()
Iterates the BioPax XML and calls the appropriate methods for each element. This method should not be called directly, `BioPaxReader.process()` calls it.

open_biopax()
Opens the BioPax file. This method should not be called directly, `BioPaxReader.process()` calls it.

process(*silent=False*)
This method executes the total workflow of BioPax processing.

Parameters **silent** (*bool*) – whether to print status messages and progress bars.

set_progress()
Initializes a progress bar. This method should not be called directly, `BioPaxReader.process()` calls it.

2.34 reflists

pypath.utils.reflists.**check**(*name*, *id_type*, *ncbi_tax_id=None*)
Checks if the identifier `name` is in the reference list with the provided `id_type` and organism.

pypath.utils.reflists.**is_not**(*names*, *id_type*, *ncbi_tax_id=None*)
Returns the identifiers from `names` which are not instances of the provided `id_type` and from the given organism.

pypath.utils.reflists.**select**(*names*, *id_type*, *ncbi_tax_id=None*)
Selects the identifiers in `names` which are in the reference list with the provided `id_type` and organism.

2.35 refs

pypath.internals.refs.**get_pmid**(*idList*)
For a list of doi or PMC IDs fetches the corresponding PMIDs.

pypath.internals.refs.**get_pubmed_data**(*pp*, *cachefile=None*, *htp_threshold=20*)
For one PyPath object, obtains metadata for all PubMed IDs through NCBI E-utils.

Parameters

- **pp** – `pypath.PyPath` object
- **htp_threshold** – The number of interactions for one reference above the study considered to be high-throughput.

pypath.internals.refs.**only_pmids**(*idList*, *strict=True*)
Return elements unchanged which comply with PubMed ID format, and attempts to translate the DOIs and PMC IDs using NCBI E-utils. Returns list containing only PMIDs.

@idList [list, str] List of IDs or one single ID.

@strict [bool] Whether keep in the list those IDs which are not PMIDs, neither DOIs or PMC IDs or NIH manuscript IDs.

pypath.internals.refs.**open_pubmed**(*pmid*)
Opens PubMed record in web browser.

@pmid [str or int] PubMed ID

2.36 residues

class pypath.utils.residues.**ResidueMapper**

This class stores and serves the PDB –> UniProt residue level mapping. Attempts to download the mapping, and stores it for further use. Converts PDB residue numbers to the corresponding UniProt ones.

clean()

Removes cached mappings, freeing up memory.

2.37 resource

class pypath.internals.resource.**AbstractResource**(*name*, *ncbi_tax_id=9606*, *input_method=None*, *input_args=None*, *dump=None*, *data_attr_name=None*, ***kwargs*)

Generic class for downloading, processing and serving data from a resource.

load_data()

Loads the data by calling input_method.

process()

Calls the _process_method.

set_method()

Sets the data input method by looking up in inputs module if necessary.

class pypath.internals.resource.**EnzymeSubstrateResourceKey**(*name*, *data_type*, *via*)

data_type

Alias for field number 1

name

Alias for field number 0

via

Alias for field number 2

class pypath.internals.resource.**NetworkResourceKey**(*name*, *data_type*, *interaction_type*, *data_model*, *via*)

data_model

Alias for field number 3

data_type

Alias for field number 1

interaction_type

Alias for field number 2

name

Alias for field number 0

via

Alias for field number 4

2.38 seq

pypath.utils.seq.**get_isoforms**(*organism=9606*)
Loads UniProt sequences for all isoforms.

pypath.utils.seq.**read_fasta**(*fasta*)
Parses a fasta file. Returns dict with headers as keys and sequences as values.

pypath.utils.seq.**swissprot_seq**(*organism=9606*, *isoforms=False*)
Loads all sequences for an organism, optionally for all isoforms, by default only first isoform.

2.39 server

2.40 session

pypath.share.session.**get_log**()
Returns the `log.Logger` instance belonging to the session.

pypath.share.session.**get_session**()
Creates new session or returns the one already created.

pypath.share.session.**new_session**(*label=None*, *log_verbosity=0*)
Creates a new session. In case one already exists it will be deleted.

label [str] A custom name for the session.

log_verbosity [int] Verbositiy level passed to the logger.

2.41 settings

class pypath.share.settings.**Defaults**(*acsn*, *acsn_names*, *alzpw_ppi*, *annotations_mod*, *annotations_pickle*, *arn*, *basedir*, *cachedir*, *complex_mod*, *complex_pickle*, *console_verbosity*, *curated_mod*, *curated_pickle*, *data_basedir*, *datasets*, *deathdomain*, *default_name_types*, *default_organism*, *dependencies*, *dorothea_expand_levels*, *enz_sub_mod*, *enz_sub_pickle*, *figures_dir*, *go_pickle_cache*, *go_pickle_cache_fname*, *goose_ancest_sql*, *goose_annot_sql*, *goose_terms_sql*, *hpmr_preprocessed*, *intercell_mod*, *intercell_pickle*, *keep_noref*, *latex_dir*, *lmpid*, *lncrna_mrna_mod*, *lncrna_mrna_pickle*, *log_flush_interval*, *log_verbosity*, *mapper_cleanup_interval*, *mapping_use_cache*, *mirna_mrna_mod*, *mirna_mrna_pickle*, *module_name*, *msigdb_email*, *nci_pid*, *network_expand_complexes*, *network_extra_directions*, *network_keep_original_names*, *network_pickle_cache*, *nrf2ome*, *old_dbptm*, *omnipath_args*, *omnipath_mod*, *omnipath_pickle*, *path_root*, *pickle_dir*, *ppoint*, *progressbars*, *pubmed_cache*, *slk01human*, *slk3_edges*, *slk3_nodes*, *tables_dir*, *tf_mirna_mod*, *tf_mirna_pickle*, *tf_target_mod*, *tf_target_pickle*, *tfregulons_levels*, *timestamp_dirs*, *timestamp_format*, *trip_preprocessed*, *uniprot_uploadlists_chunk_size*, *use_intermediate_cache*, *webpage_main*)

acsn
Alias for field number 0

acsn_names
Alias for field number 1

alzpw_ppi
Alias for field number 2

annotations_mod
Alias for field number 3

annotations_pickle
Alias for field number 4

arn
Alias for field number 5

basedir
Alias for field number 6

cachedir
Alias for field number 7

complex_mod
Alias for field number 8

complex_pickle
Alias for field number 9

console_verbosity
Alias for field number 10

curated_mod
Alias for field number 11

curated_pickle
Alias for field number 12

data_basedir
Alias for field number 13

datasets
Alias for field number 14

deathdomain
Alias for field number 15

default_name_types
Alias for field number 16

default_organism
Alias for field number 17

dependencies
Alias for field number 18

dorothea_expand_levels
Alias for field number 19

enz_sub_mod
Alias for field number 20

enz_sub_pickle
Alias for field number 21

figures_dir
Alias for field number 22

go_pickle_cache
Alias for field number 23

go_pickle_cache_fname
Alias for field number 24

goose_ancest_sql
Alias for field number 25

goose_annot_sql
Alias for field number 26

goose_terms_sql
Alias for field number 27

hpmr_preprocessed
Alias for field number 28

intercell_mod
Alias for field number 29

intercell_pickle
Alias for field number 30

keep_noref
Alias for field number 31

latex_dir
Alias for field number 32

lmpid
Alias for field number 33

lncrna_mrna_mod
Alias for field number 34

lncrna_mrna_pickle
Alias for field number 35

log_flush_interval
Alias for field number 36

log_verbosity
Alias for field number 37

mapper_cleanup_interval
Alias for field number 38

mapping_use_cache
Alias for field number 39

mirna_mrna_mod
Alias for field number 40

mirna_mrna_pickle
Alias for field number 41

module_name
Alias for field number 42

msigdb_email
Alias for field number 43

nci_pid
Alias for field number 44

network_expand_complexes
Alias for field number 45

network_extra_directions
Alias for field number 46

network_keep_original_names
Alias for field number 47

network_pickle_cache
Alias for field number 48

nrf2ome
Alias for field number 49

old_dbptm
Alias for field number 50

omnipath_args
Alias for field number 51

omnipath_mod
Alias for field number 52

omnipath_pickle
Alias for field number 53

path_root
Alias for field number 54

pickle_dir
Alias for field number 55

ppoint
Alias for field number 56

progressbars
Alias for field number 57

pubmed_cache
Alias for field number 58

slk01human
Alias for field number 59

slk3_edges
Alias for field number 60

slk3_nodes
Alias for field number 61

tables_dir
Alias for field number 62

tf_mirna_mod
Alias for field number 63

tf_mirna_pickle
Alias for field number 64

tf_target_mod
Alias for field number 65

tf_target_pickle
Alias for field number 66

tfregulons_levels
Alias for field number 67

timestamp_dirs
Alias for field number 68

timestamp_format
Alias for field number 69

trip_preprocessed
Alias for field number 70

uniprot_uploadlists_chunk_size
Alias for field number 71

use_intermediate_cache
Alias for field number 72

webpage_main
Alias for field number 73

2.42 taxonomy

2.43 unichem

2.44 uniprot

2.45 urls

2.46 build

This is a standalone module with the only purpose of building the tables for the webservice.

class pypath.omnipath.server.build.**WebserviceTables**(*only_human=False*, *outfile_interactions='omnipath_webservice_interactions.tsv'*, *outfile_ptms='omnipath_webservice_enz_sub.tsv'*, *outfile_complexes='omnipath_webservice_complexes.tsv'*, *outfile_annotations='omnipath_webservice_annotations.tsv'*, *outfile_intercell='omnipath_webservice_intercell.tsv'*, *network_datasets=None*)
Creates the data frames which the web service uses to serve the data from.

2.47 resources

2.47.1 network

pypath.resources.network.**dorothea_expand_levels**(*resources=None*, *levels=None*)
In a dictionary of resource definitions creates a separate `NetworkResource` object for each confidence levels of DoRothEA just like each level was a different resource.

No matter `resources` is a `NetworkResource` or a dict of network resources, returns always a dict of network resources.

CHAPTER

THREE

WEBSERVICE

New webservice from 14 June 2018: the queries slightly changed, have been largely extended. See the examples below.

One instance of the pypath webservice runs at the domain http://omnipathdb.org/, serving not only the OmniPath data but other datasets: TF-target interactions from TF Regulons, a large collection additional enzyme-substrate interactions, and literature curated miRNA-mRNA interacions combined from 4 databases. The webservice implements a very simple REST style API, you can make requests by HTTP protocol (browser, wget, curl or whatever).

The webservice currently recognizes 3 types of queries: `interactions`, `ptms` and `info`. The query types `resources`, `network` and `about` have not been implemented yet in the new webservice.

3.1 Mouse and rat

Except the miRNA interactions all interactions are available for human, mouse and rat. The rodent data has been translated from human using the NCBI Homologene database. Many human proteins have no known homolog in rodents hence rodent datasets are smaller than their human counterparts. Note, if you work with mouse omics data you might do better to translate your dataset to human (for example using the `pypath.homology` module) and use human interaction data.

3.2 Examples

A request without any parameter, gives some basic numbers about the actual loaded dataset:

> http://omnipathdb.org

The `info` returns a HTML page with comprehensive information about the resources:

> http://omnipathdb.org/info

The `interactions` query accepts some parameters and returns interactions in tabular format. This example returns all interactions of EGFR (P00533), with sources and references listed.

> http://omnipathdb.org/interactions/?partners=P00533&fields=sources,references

By default only the OmniPath dataset used, to query the TF Regulons or add the extra enzyme-substrate interactions you need to set additional parameters. For example to query the transcriptional regulators of EGFR:

> http://omnipathdb.org/interactions/?targets=EGFR&types=TF

The TF Regulons database assigns confidence levels to the interactions. You might want to select only the highest confidence, *A* category:

> http://omnipathdb.org/interactions/?targets=EGFR&types=TF&tfregulons_levels=A

Show the transcriptional targets of Smad2 homology translated to rat including the confidence levels from TF Regulons:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=type,ncbi_tax_id,tfregulons_level&organisms=10116&sources=Smad2&types=TF

Query interactions from PhosphoNetworks which is part of the *kinaseextra* dataset:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources&databases=PhosphoNetworks&datasets=kinaseextra

Get the interactions from Signor, SPIKE and SignaLink3:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&databases=Signor,SPIKE,SignaLink3

All interactions of MAP1LC3B:

> http://omnipathdb.org/interactions/?genesymbols=1&partners=MAP1LC3B

By default `partners` queries the interaction where either the source or the arget is among the partners. If you set the `source_target` parameter to `AND` both the source and the target must be in the queried set:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&sources=ATG3,ATG7,ATG4B,SQSTM1&targets=MAP1LC3B,MAP1LC3A,MAP1LC3C,Q9H0R8,GABARAP,GABARAPL2&source_target=AND

As you see above you can use UniProt IDs and Gene Symbols in the queries and also mix them. Get the miRNA regulating NOTCH1:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&datasets=mirnatarget&targets=NOTCH1

Note: with the exception of mandatory fields and genesymbols, the columns appear exactly in the order you provided in your query.

Another query type available is `ptms` which provides enzyme-substrate interactions. It is very similar to the `interactions`:

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references,isoforms&enzymes=FYN

Is there any ubiquitination reaction?

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&types=ubiquitination

And acetylation in mouse?

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&types=acetylation&organisms=10090

Rat interactions, both directly from rat and homology translated from human, from the PhosphoSite database:

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&organisms=10116&databases=PhosphoSite,PhosphoSite_noref

CHAPTER

FOUR

RELEASE HISTORY

Main improvements in the past releases:

4.1 0.1.0

- First release of pypath, for initial testing.

4.2 0.2.0

- Lots of small improvements in almost every module
- Networks can be read from local files, remote files, lists or provided by any function
- Almost all redistributed data have been removed, every source downloaded from the original provider.

4.3 0.3.0

- First version with partial Python 3 support.

4.4 0.4.0

- **pyreact** module with **BioPaxReader** and **PyReact** classes added
- Process description databases, BioPax and PathwayCommons SIF conversion rules are supported
- Format definitions for 6 process description databases included.

4.5 0.5.0

- Many classes have been added to the **plot** module
- All figures and tables in the manuscript can be generated automatically
- This is supported by a new module, **analysis**, which implements a generic workflow in its **Workflow** class.

4.6 0.7.74

- **homology** module: finds the homologs of proteins using the NCBI Homologene database and the homologs of PTM sites using UniProt sequences and PhosphoSitePlus homology table
- **ptm** module: fully integrated way of processing enzyme-substrate interactions from many databases and their translation by homology to other species
- **export** module: creates `pandas.DataFrame` or exports the network into tabular file
- New webservice
- TF Regulons database included and provides much more comprehensive transcriptional regulation resources, including literature curated, in silico predicted, ChIP-Seq and expression pattern based approaches
- Many network resources added, including miRNA-mRNA and TF-miRNA interactions

4.7 Upcoming

- New, more flexible network reader class
- Full support for multi-species molecular interaction networks (e.g. pathogene-host)
- Better support for not protein only molecular interaction networks (metabolites, drug compounds, RNA)
- ChEMBL webservice interface, interface for PubChem and eventually forDrugBank
- Silent mode: a way to suppress messages and progress bars

pypath consists of a number of submodules to build various databases. Most of these are provided as **pandas** data frames. The network database is built around igraph to work with molecular network representations e.g. protein, miRNA and drug compound interaction networks.

CHAPTER

FIVE

WEBSERVICE

New webservice from 14 June 2018: the queries slightly changed, have been largely extended. See the examples below.

The webservice implements a very simple REST style API, you can make requests by the HTTP protocol (browser, wget, curl or whatever). After defining the query type and optionally a set of molecular entities (proteins) you can add further GET parameters encoded in the URL.

5.1 Query types

The webservice currently recognizes 7 types of queries: `interactions`, `ptms`, `annotations`, `complexes`, `intercell`, `queries` and `info`. The query types `resources`, `network` and `about` have not been implemented yet in the new webservice.

5.2 Interaction datasets

The instance of the `pypath` webserver running at the domain http://omnipathdb.org/, serves not only the OmniPath data but also other datasets. Each of them has a short name what you can use in the queries (e.g. `&datasets=omnipath,pathwayextra`).

- `omnipath`: the OmniPath data as defined in the paper, an arbitrary optimum between coverage and quality
- `pathwayextra`: activity flow interactions without literature reference
- `kinaseextra`: enzyme-substrate interactions without literature reference
- `ligrecextra`: ligand-receptor interactions without literature reference
- `tfregulons`: transcription factor (TF)-target interactions from DoRothEA
- `mirnatarget`: miRNA-mRNA and TF-miRNA interactions

TF-target interactions from TF Regulons, a large collection additional enzyme-substrate interactions, and literature curated miRNA-mRNA interacions combined from 4 databases.

5.3 Mouse and rat

Except the miRNA interactions all interactions are available for human, mouse and rat. The rodent data has been translated from human using the NCBI Homologene database. Many human proteins do not have known homolog in rodents hence rodent datasets are smaller than their human counterparts. Note, if you work with mouse omics data

you might do better to translate your dataset to human (for example using the `pypath.homology` module) and use human interaction data.

5.4 Examples

A request without any parameter provides the main webpage:

http://omnipathdb.org

The `info` returns a HTML page with comprehensive information about the resources. The list here should be and will be updated as currently OmniPath includes much more databases:

http://omnipathdb.org/info

5.4.1 Molecular interaction network

The `interactions` query accepts some parameters and returns interactions in tabular format. This example returns all interactions of EGFR (P00533), with sources and references listed.

http://omnipathdb.org/interactions/?partners=P00533&fields=sources,references

By default only the OmniPath dataset used, to include any other dataset you have to set additional parameters. For example to query the transcriptional regulators of EGFR:

http://omnipathdb.org/interactions/?targets=EGFR&types=TF

The TF Regulons database assigns confidence levels to the interactions. You might want to select only the highest confidence, *A* category:

http://omnipathdb.org/interactions/?targets=EGFR&types=TF&tfregulons_levels=A

Show the transcriptional targets of Smad2 homology translated to rat including the confidence levels from TF Regulons:

http://omnipathdb.org/interactions/?genesymbols=1&fields=type,ncbi_tax_id,tfregulons_level&organisms=10116&sources=Smad2&types=TF

Query interactions from PhosphoNetworks which is part of the *kinaseextra* dataset:

http://omnipathdb.org/interactions/?genesymbols=1&fields=sources&databases=PhosphoNetworks&datasets=kinaseextra

Get the interactions from Signor, SPIKE and SignaLink3:

http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&databases=Signor,SPIKE,SignaLink3

All interactions of MAP1LC3B:

http://omnipathdb.org/interactions/?genesymbols=1&partners=MAP1LC3B

By default `partners` queries the interaction where either the source or the arget is among the partners. If you set the `source_target` parameter to `AND` both the source and the target must be in the queried set:

http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&sources=ATG3,ATG7,ATG4B,SQSTM1&targets=MAP1LC3B,MAP1LC3A,MAP1LC3C,Q9H0R8,GABARAP,GABARAPL2&source_target=AND

As you see above you can use UniProt IDs and Gene Symbols in the queries and also mix them. Get the miRNA regulating NOTCH1:

http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&datasets=mirnatarget&targets=NOTCH1

Note: with the exception of mandatory fields and genesymbols, the columns appear exactly in the order you provided in your query.

5.4.2 Enzyme-substrate interactions

Another query type available is `ptms` which provides enzyme-substrate interactions. It is very similar to the `interactions`:

http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references,isoforms&enzymes=FYN

Is there any ubiquitination reaction?

http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&types=ubiquitination

And acetylation in mouse?

http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&types=acetylation&organisms=10090

Rat interactions, both directly from rat and homology translated from human, from the PhosphoSite database:

http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&organisms=10116&databases=PhosphoSite,PhosphoSite_noref

5.4.3 Molecular complexes

The `complexes` query provides a comprehensive database of more than 22,000 protein complexes. For example, to query all complexes from CORUM and PDB containing MTOR (P42345):

http://omnipathdb.org/complexes?proteins=P42345&databases=CORUM,PDB

5.4.4 Annotations

The `annotations` query provides a large variety of data about proteins, complexes and in the future other kinds of molecules. For example an annotation can tell if a protein is a kinase, or if it is expressed in the hearth muscle. These data come from dozens of databases and each kind of annotation record contains different fields. Because of this here we have a `record_id` field which is unique within the records of each database. Each row contains one key value pair and you need to use the `record_id` to connect the related key-value pairs. You can easily do this with `tidyr` and `dplyr` in R or `pandas` in Python. An example to query the pathway annotations from SignaLink:

http://omnipathdb.org/annotations?databases=SignaLink3

Or the tissue expression of BMP7 from Human Protein Atlas:

http://omnipathdb.org/annotations?databases=HPA&proteins=BMP7

5.4.5 Roles in inter-cellular communication

Another query type is `intercell` providing information on the roles in inter-cellular signaling. E.g. if a protein is a ligand, a receptor, an extracellular matrix (ECM) component, etc. This query type is very similar to `annotations` but here the data does not come from original sources but combined from several databases by us. However we refer also to the original databases whenever the `class_type` is `sub` (subclass). E.g. the main class `ligand` is a combination of `Ramilowski 2015`, `CellPhoneDB`, `HPMR` and many other databases, hence besides the

`ligand` category you will find sub-categories like `ligand_ramilowski`, `ligand_cellphonedb` and so on. An example how to get all intercell annotations for 4 selected proteins:

http://omnipathdb.org/intercell?proteins=EGFR,ULK1,ATG4A,BMP8B

Or all the main classes for one protein:

http://omnipathdb.org/intercell?levels=main&proteins=P00533

Or a list of all ECM proteins:

http://omnipathdb.org/intercell?categories=ecm

5.4.6 Exploring possible parameters

Sometimes the names and values of the query parameters are not intuitive, even though in many cases the server accepts multiple alternatives. To see the possible parameters with all possible values you can use the `queries` query type. The server checks the paremeter names and values exactly against these rules and if any of them don't match you will get an error message instead of reply. To see the parameters for the `interactions` query:

http://omnipathdb.org/queries/interactions

CHAPTER

SIX

CAN I USE OMNIPATH IN R?

You can download the data from the webservice and load into R. Thanks to our colleague Attila Gabor we have a dedicated package for this:

https://github.com/saezlab/OmnipathR

Alternatively here is a very simple example:

https://github.com/saezlab/pypath/tree/master/r_import

CHAPTER

SEVEN

INSTALLATION

7.1 Linux

In almost any up-to-date Linux distribution the dependencies of **pypath** are built-in, or provided by the distributors. You only need to install a couple of things in your package manager (cairo, py(2)cairo, igraph, python(2)-igraph, graphviz, pygraphviz), and after install **pypath** by *pip* (see below). If any module still missing, you can install them the usual way by *pip* or your package manager.

7.2 igraph C library, cairo and pycairo

python(2)-igraph is a Python interface to use the igraph C library. The C library must be installed. The same goes for *cairo*, *py(2)cairo* and *graphviz*.

7.3 Directly from git

```
pip install git+https://github.com/saezlab/pypath.git
```

7.4 With pip

Download the package from /dist, and install with pip:

```
pip install pypath-x.y.z.tar.gz
```

7.5 Build source distribution

Clone the git repo, and run setup.py:

```
python setup.py sdist
```

7.6 Mac OS X

On OS X installation is not straightforward primarily because cairo needs to be compiled from source. We provide 2 scripts here: the **mac-install-brew.sh** installs everything with HomeBrew, and **mac-install-conda.sh** installs from Anaconda distribution. With these scripts installation of igraph, cairo and graphviz goes smoothly most of the time, and options are available for omitting the 2 latter. To know more see the description in the script header. There is a third script **mac-install-source.sh** which compiles everything from source and presumes only Python 2.7 and Xcode installed. We do not recommend this as it is time consuming and troubleshooting requires expertise.

7.6.1 Troubleshooting

- `no module named ...` when you try to load a module in Python. Did theinstallation of the module run without error? Try to run again the specific part from the mac install shell script to see if any error comes up. Is the path where the module has been installed in your `$PYTHONPATH`? Try `echo $PYTHONPATH` to see the current paths. Add your local install directories if those are not there, e.g. `export PYTHONPATH="/Users/me/local/python2.7/site-packages:$PYTHONPATH"`. If it works afterwards, don't forget to append these export path statements to your `~/.bash_profile`, so these will be set every time you launch a new shell.
- `pkgconfig` not found. Check if the `$PKG_CONFIG_PATH` variable is set correctly, and pointing on a directory where pkgconfig really can be found.
- Error while trying to install py(2)cairo by pip. py(2)cairo could not be installed by pip, but only by waf. Please set the `$PKG_CONFIG_PATH` before. See **mac-install-source.sh** on how to install with waf.
- Error at pygraphviz build: `graphviz/cgraph.h file not found`. This is because the directory of graphviz detected wrong by pkgconfig. See **mac-install-source.sh** how to set include dirs and library dirs by `--global-option` parameters.
- Can not install bioservices, because installation of jurko-suds fails. Ok, this fails because pip is not able to install the recent version of setuptools, because a very old version present in the system path. The development version of jurko-suds does not require setuptools, so you can install it directly from git as it is done in **mac-install-source.sh**.
- In **Anaconda**, *pypath* can be imported, but the modules and classes are missing. Apparently Anaconda has some built-in stuff called *pypath*. This has nothing to do with this module. Please be aware that Anaconda installs a completely separated Python distribution, and does not detect modules in the main Python installation. You need to install all modules within Anaconda's directory. **mac-install-conda.sh** does exactly this. If you still experience issues, please contact us.

7.7 Microsoft Windows

Not many people have used *pypath* on Microsoft computers so far. Please share your experiences and contact us if you encounter any issue. We appreciate your feedback, and it would be nice to have better support for other computer systems.

7.7.1 With Anaconda

The same workflow like you see in `mac-install-conda.sh` should work for Anaconda on Windows. The only problem you certainly will encounter is that not all the channels have packages for all platforms. If certain channel provides no package for Windows, or for your Python version, you just need to find an other one. For this, do a search:

```
anaconda search -t conda <package name>
```

For example, if you search for *pycairo*, you will find out that *vgauther* provides it for osx-64, but only for Python 3.4, while *richlewis* provides also for Python 3.5. And for win-64 platform, there is the channel of *KristanAmstrong*. Go along all the commands in `mac-install-conda.sh`, and modify the channel if necessary, until all packages install successfully.

7.7.2 With other Python distributions

Here the basic principles are the same as everywhere: first try to install all external dependencies, after *pip* install should work. On Windows certain packages can not be installed by compiled from source by *pip*, instead the easiest to install them precompiled. These are in our case *fisher, lxml, numpy (mkl version), pycairo, igraph, pygraphviz, scipy and statsmodels*. The precompiled packages are available here: http://www.lfd.uci.edu/~gohlke/pythonlibs/. We tested the setup with Python 3.4.3 and Python 2.7.11. The former should just work fine, while with the latter we have issues to be resolved.

7.7.3 Known issues

- *"No module fabric available."* – or *pysftp* missing: this is not important, only certain data download methods rely on these modules, but likely you won't call those at all.
- Progress indicator floods terminal: sorry about that, will be fixed soon.
- Encoding related exceptions in Python2: these might occur at some points in the module, please send the traceback if you encounter one, and we will fix as soon as possible.
- For Mac OS X (v >= 10.11 El Capitan) import of pypath fails with error: "libcurl link-time ssl backend (openssl) is different from compile-time ssl backend (none/other)". To fix it, you may need to reinstall pycurl library using special flags. More information and steps can be found e.g. [here](https://cscheng.info/2018/01/26/installing-pycurl-on-macos-high-sierra.html)

Special thanks to Jorge Ferreira for testing pypath on Windows!

CHAPTER

EIGHT

RELEASE HISTORY

Main improvements in the past releases:

8.1 0.1.0

- First release of PyPath, for initial testing.

8.2 0.2.0

- Lots of small improvements in almost every module
- Networks can be read from local files, remote files, lists or provided by any function
- Almost all redistributed data have been removed, every source downloaded from the original provider.

8.3 0.3.0

- First version with partial Python 3 support.

8.4 0.4.0

- **pyreact** module with **BioPaxReader** and **PyReact** classes added
- Process description databases, BioPax and PathwayCommons SIF conversion rules are supported
- Format definitions for 6 process description databases included.

8.5 0.5.0

- Many classes have been added to the **plot** module
- All figures and tables in the manuscript can be generated automatically
- This is supported by a new module, **analysis**, which implements a generic workflow in its **Workflow** class.

8.6 0.5.32

- *chembl*, *unichem*, *mysql* and *mysql_connect* modules made Python3 compatible

8.7 0.6.31

- Orthology translation of network
- Homologene UniProt dict to translate between different organisms UniProt-to-UniProt
- Orthology translation of PTMs
- Better processing of PhosphoSite regulatory sites

8.8 0.7.0

- TF-target, miRNA-mRNA and TF-miRNA interactions from many databases

8.9 0.7.74

- New web server based on *pandas* data frames
- New module *export* for generating data frames of interactions or enzyme-substrate interactions
- New module *websrvtab* for exporting data frames for the web server
- TF-target interactions from DoRothEA

8.10 0.7.93

- New *dataio* methods for Gene Ontology

8.11 0.7.110

- Many new docstrings

8.12 0.8

- New module *complex*: a comprehensive database of complexes
- New module *annot*: database of protein annotations (function, location)
- New module *intercell*: special methods for data integration focusing on intercellular communication
- New module *bel*: BEL integration
- Module *go* and all the connected *dataio* methods have been rewritten offering a workaround for data access despite GO's terrible web services and providing much more versatile query methods

- Removed MySQL support (e.g. loading mapping tables from MySQL)
- Modules *mapping*, *reflists*, *complex*, *ptm*, *annot*, *go* became services: these modules build databases and provide query methods, sometimes they even automatically delete data to free memory
- New interaction category in *data_formats*: *ligand_receptor*
- Improved logging and control over verbosity
- Better control over paremeters by the *settings* module
- Many methods in *dataio* have been improved or fixed, docs and code style largely improved
- Started to add tests especially for methods in *dataio*

8.13 0.9

- The network database is not dependent any more on *python-igraph* hence it has been removed from the mandatory dependencies
- New API for the network, interactions, evidences, molecular entities

8.14 Upcoming

- New, more flexible network reader class
- Full support for multi-species molecular interaction networks (e.g. pathogene-host)
- Better support for not protein only molecular interaction networks (metabolites, drug compounds, RNA)

CHAPTER

NINE

FEATURES

In the beginning the primary aim of **pypath** was to build networks from multiple sources using an igraph object as the fundament of the integrated data structure. From version 0.7 and 0.8 this design principle started to change. Today **pypath** builds a number of different databases each having **pandas.DataFrame** as a final format. Each of these integrates a specific kind of data from various databases (e.g. protein complexes, interactions, enzyme-PTM relationships, etc). **pypath** has many submodules with standalone functionality which can be used in other modules and scripts. For example the ID conversion module **pypath.mapping**.

Submodules perform various features, e.g. graph visualization, working with rug compound data, searching drug targets and compounds in **ChEMBL**.

9.1 ID conversion

The ID conversion module `mapping` can be used independently. It has the feature to translate secondary UniProt IDs to primaries, and Trembl IDs to SwissProt, using primary Gene Symbols to find the connections. This module automatically loads and stores the necessary conversion tables. Many tables are predefined, such as all the IDs in **UniProt mapping service,** while users are able to load any table from **file** or **MySQL,** using the classes provided in the module `input_formats`.

9.2 Pathways

pypath includes data and predefined format descriptions for more than 25 high quality, literature curated databases. The inut formats are defined in the `data_formats` module. For some resources data downloaded on the fly, where it is not possible, data is redistributed with the module. Descriptions and comprehensive information about the resources is available in the `descriptions` module.

9.3 Structural features

One of the modules called `intera` provides many classes for representing structures and mechanisms behind protein interactions. These are `Residue` (optionally mutated), `Motif`, `Ptm`, `Domain`, `DomainMotif`, `DomainDomain` and `Interface`. All these classes have `__eq__()` methods to test equality between instances, and also `__contains__()` methods to look up easily if a residue is within a short motif or protein domain, or is the target residue of a PTM.

9.4 Sequences

The module `seq` contains a simple class for quick lookup any residue or segment in **UniProt** protein sequences while being aware of isoforms.

9.5 Tissue expression

For 3 protein expression databases there are functions and modules for downloading and combining the expression data with the network. These are the Human Protein Atlas, the ProteomicsDB and GIANT. The `giant` and `proteomicsdb` modules can be used also as stand alone Python clients for these resources.

9.6 Functional annotations

GSEA and **Gene Ontology** are two approaches for annotating genes and gene products, and enrichment analysis technics aims to use these annotations to highlight the biological functions a given set of genes is related to. Here the `enrich` module gives abstract classes to calculate enrichment statistics, while the `go` and the `gsea` modules give access to GO and GSEA data, and make it easy to count enrichment statistics for sets of genes.

9.7 Drug compounds

UniChem submodule provides an interface to effectively query the UniChem service, use connectivity search with custom settings, and translate SMILEs to ChEMBL IDs with ChEMBL web service.

ChEMBL submodule queries directly your own ChEMBL MySQL instance, has the features to search targets and compounds from custom assay types and relationship types, to get activity values, binding domains, and action types. You need to download the ChEMBL MySQL dump, and load into your own server.

9.8 Technical

The module `pypath.curl` provides a very flexible **download manager** built on top of `pycurl`. The classes `pypath.curl.Curl()` and `pypath.curl.FileOpener` accept numerous arguments, try to deal in a smart way with local **cache,** authentication, redirects, uncompression, character encodings, FTP and HTTP transactions, and many other stuff. Cache can grow to several GBs, and takes place in `~/.pypath/cache` by default. If you experience issues using `pypath` these are most often related to failed downloads which often result nonsense cache contents. To debug such issues you can see the cache file names and cache usage in the log, and you can use the context managers in `pypath.curl` to show, delete or bypass the cache for some particular method calls (`pypath.curl.cache_print_on()`, `pypath.curl.cache_delete_on()` and `pypath.curl.cache_off()`. You can always set up an alternative cache directory for the entire session using the `pypath.settings` module.

The `pypath.session` and `pypath.log` modules take care of setting up session level parameters and logging. Each session has a random 5 character identifier e.g. `y5jzx`. The default log file in this case is `pypath_log/pypath-y5jzx.log`. The log messages flushed in every 2 seconds by default. You can always change these things by the `settings` module. In this module you can get and set the values of various parameters using the `pypath.settings.setup()` and the `pypath.settings.get()` methods.

A simple **webservice** comes with this module: the `server` module based on `twisted.web.server` opens a custom port and serves plain text tables over HTTP with REST style querying.

PYTHON MODULE INDEX

p

INDEX

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z