
PyBEL Documentation

Release 0.15.4

Charles Tapley Hoyt

Jul 07, 2021

GETTING STARTED

1	Overview	3
1.1	Background on Systems Biology Modeling	3
1.2	Design Considerations	3
1.3	Implementation	4
1.4	Extensions to BEL	4
1.5	Things to Consider	5
2	Installation	9
2.1	Extras	9
2.2	Caveats	10
2.3	Upgrading	10
3	Data Model	11
3.1	Constants	11
3.2	Graph	12
3.3	Nodes	25
3.4	Unqualified Edges	32
3.5	Edges	34
4	Example Networks	39
5	Filters	43
6	Grouping	53
7	Operations	55
8	Pipeline	57
8.1	Transformation Decorators	59
8.2	Exceptions	60
9	Query	61
10	Summary	65
11	Mutations	71
12	Collapse	73
13	Deletion	75
14	Expansion	77

15 Induction	79
16 Induction and Expansion	83
17 Inference	85
18 Metadata	87
19 Input and Output	89
19.1 Import	90
19.2 Transport	92
19.3 Visualization	101
19.4 Analytical Services	102
19.5 Web Services	107
19.6 Databases	113
19.7 Lossy Export	114
20 Manager	117
20.1 Manager API	117
20.2 Manager Components	119
21 Models	123
22 Cookbook	129
22.1 Configuration	129
23 Command Line Interface	131
23.1 pybel	131
24 Constants	139
25 Parsers	151
25.1 BEL Parser	151
25.2 Metadata Parser	155
25.3 Control Parser	158
25.4 Concept Parser	160
25.5 Sub-Parsers	162
26 Internal Domain Specific Language	163
26.1 Primitives	163
26.2 Named Entities	164
26.3 Central Dogma	165
26.4 Fusions	170
26.5 Utilities	173
27 Logging Messages	175
28 References	185
28.1 Related Publications	185
28.2 Software using PyBEL	185
28.3 BEL Content	186
29 Current Issues	187
29.1 Speed	187
29.2 Namespaces	187
29.3 Testing	187

30 Technology	189
30.1 Versioning	189
30.2 Testing in PyBEL	189
30.3 Deployment	190
31 Indices and Tables	193
Bibliography	195
Python Module Index	197
Index	199

Biological Expression Language (BEL) is a domain-specific language that enables the expression of complex molecular relationships and their context in a machine-readable form. Its simple grammar and expressive power have led to its successful use in the to describe complex disease networks with several thousands of relationships.

PyBEL is a pure Python software package that parses BEL documents, validates their semantics, and facilitates data interchange between common formats and database systems like JSON, CSV, Excel, SQL, CX, and Neo4J. Its companion package, [PyBEL-Tools](#), contains a library of functions for analysis of biological networks. For result-oriented guides, see the [PyBEL-Notebooks](#) repository.

Installation is as easy as getting the code from [PyPI](#) with `python3 -m pip install pybel`. See the [installation](#) documentation.

For citation information, see the [references](#) page.

PyBEL is tested on Python 3.5+ on Mac OS and Linux using [Travis CI](#) as well as on Windows using [AppVeyor](#).

See also:

- Specified by [BEL 1.0](#), [BEL 2.0](#), and [BEL 2.0+](#)
- Documented on [Read the Docs](#)
- Versioned on [GitHub](#)
- Tested on [Travis CI](#)
- Distributed by [PyPI](#)

OVERVIEW

1.1 Background on Systems Biology Modeling

1.1.1 Biological Expression Language (BEL)

Biological Expression Language (BEL) is a domain specific language that enables the expression of complex molecular relationships and their context in a machine-readable form. Its simple grammar and expressive power have led to its successful use to describe complex disease networks with several thousands of relationships. For a detailed explanation, see the BEL 1.0 and 2.0, and 2.0+ specifications.

1.1.2 BEL Community Links

- [BEL Community Portal](#)
- [BEL Google Group](#)

1.2 Design Considerations

1.2.1 Missing Namespaces and Improper Names

The use of openly shared controlled vocabularies (namespaces) within BEL facilitates the exchange and consistency of information. Finding the correct `namespace:name` pair is often a difficult part of the curation process.

1.2.2 Outdated Namespaces

BEL provides a variety of [namespaces](#) covering each of the BEL function types. Selventa used to provide BEL namespace files generated by the deprecated project at <https://github.com/OpenBEL/resource-generator> and hosted at the abandoned website <http://www.belframework.org/>. Newer versions of these namespaces can be found at <https://github.com/pharmacome/conso/tree/master/external>.

1.2.3 Generating New Namespaces

In some cases, it is appropriate to design a new namespace, using the [custom namespace specification](#) provided by the OpenBEL Framework. Packages for generating namespace, annotation, and knowledge resources have been grouped in the [Bio2BEL](#) organization on GitHub.

1.2.4 Synonym Issues

Due to the huge number of terms across many namespaces, it's difficult for curators to know the domain-specific synonyms that obscure the controlled/preferred term. However, the issue of synonym resolution and semantic searching has already been generally solved by the use of ontologies. Besides just a controlled vocabulary, they also a hierarchical model of knowledge, synonyms with cross-references to databases and other ontologies, and other information semantic reasoning. Ontologies in the biomedical domain can be found at [OBO](#) and [EMBL-EBI OLS](#).

Additionally, as a tool for curators, the EMBL Ontology Lookup Service (OLS) allows for semantic searching. Simple queries for the terms 'mitochondrial dysfunction' and 'amyloid beta-peptides' immediately returned results from relevant ontologies, and ended a long debate over how to represent these objects within BEL. EMBL-EBI also provides a programmatic API to the OLS service, for searching terms (<http://www.ebi.ac.uk/ols/api/search?q=folic%20acid>) and suggesting resolutions (<http://www.ebi.ac.uk/ols/api/suggest?q=folic+acid>)

1.3 Implementation

PyBEL is implemented using the PyParsing module. It provides flexibility and incredible speed in parsing compared to regular expression implementation. It also allows for the addition of parsing action hooks, which allow the graph to be checked semantically at compile-time.

It uses SQLite to provide a consistent and lightweight caching system for external data, such as namespaces, annotations, ontologies, and SQLAlchemy to provide a cross-platform interface. The same data management system is used to store graphs for high-performance querying.

1.4 Extensions to BEL

The PyBEL compiler is fully compliant with both BEL v1.0 and v2.0 and automatically upgrades legacy statements. Additionally, PyBEL includes several additions to the BEL specification to enable expression of important concepts in molecular biology that were previously missing and to facilitate integrating new data types. A short example is the inclusion of protein oxidation in the default BEL namespace for protein modifications. Other, more elaborate additions are outlined below.

1.4.1 Syntax for Epigenetics

PyBEL introduces the gene modification function, `gmod()`, as a syntax for encoding epigenetic modifications. Its usage mirrors the `pmod()` function for proteins and includes arguments for methylation.

For example, the methylation of `NDUFB6` was found to be negatively correlated with its expression in a study of insulin resistance and Type II diabetes. This can now be expressed in BEL such as in the following statement:

```
g(HGNC:NDUFB6, gmod(Me)) negativeCorrelation r(HGNC:NDUFB6)
```

References:

- <https://www.ncbi.nlm.nih.gov/pubmed/17948130>

- <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4655260/>

Note: This syntax is currently under consideration as [BEP-0006](#).

1.4.2 Definition of Namespaces as Regular Expressions

BEL imposes the constraint that each identifier must be qualified with an enumerated namespace to enable semantic interoperability and data integration. However, enumerating a namespace with potentially billions of names, such as dbSNP, poses a computational issue. PyBEL introduces syntax for defining namespaces with a consistent pattern using a regular expression to overcome this issue. For these namespaces, semantic validation can be performed in post-processing against the underlying database. The dbSNP namespace can be defined with a syntax familiar to BEL annotation definitions with regular expressions as follows:

```
DEFINE NAMESPACE dbSNP AS PATTERN "rs[0-9]+"
```

Note: This syntax was proposed with [BEP-0005](#) and has been officially accepted as part of the BEL 2.1 specification.

1.4.3 Definition of Resources using OWL

Previous versions of PyBEL until 0.11.2 had an alternative namespace definition. Now it is recommended to either generate namespace files with reproducible build scripts following the Bio2BEL framework, or to directly add them to the database with the Bio2BEL `bio2bel.manager.namespace_manager.NamespaceManagerMixin` extension.

1.5 Things to Consider

1.5.1 Do All Statements Need Supporting Text?

Yes! All statements must be minimally qualified with a citation and evidence (now called `SupportingText` in BEL 2.0) to maintain provenance. Statements without evidence can't be traced to their source or evaluated independently from the curator, so they are excluded.

1.5.2 Multiple Annotations

All single annotations are considered as single element sets. When multiple annotations are present, all are unioned and attached to a given edge.

```
SET Citation = {"PubMed", "Example Article", "12345"}
SET ExampleAnnotation1 = {"Example Value 11", "Example Value 12"}
SET ExampleAnnotation2 = {"Example Value 21", "Example Value 22"}
p(HGNC:YFG1) -> p(HGNC:YFG2)
```

1.5.3 Namespace and Annotation Name Choices

*.belns and *.belanno configuration files include an entry called “Keyword” in their respective [Namespace] and [AnnotationDefinition] sections. To maintain understandability between BEL documents, PyBEL warns when the names given in *.bel documents do not match their respective resources. For now, capitalization is not considered, but in the future, PyBEL will also warn when capitalization is not properly stylized, like forgetting the lowercase ‘h’ in “ChEMBL”.

1.5.4 Why Not Nested Statements?

BEL has different relationships for modeling direct and indirect causal relations.

Direct

- $A \Rightarrow B$ means that A directly increases B through a physical process.
- $A =| B$ means that A directly decreases B through a physical process.

Indirect

The relationship between two entities can be coded in BEL, even if the process is not well understood.

- $A \rightarrow B$ means that A indirectly increases B . There are hidden elements in X that mediate this interaction through a pathway direct interactions $A (\Rightarrow \text{ or } =|) X_1 (\Rightarrow \text{ or } =|) \dots X_n (\Rightarrow \text{ or } =|) B$, or through a set of multiple pathways that constitute a network.
- $A -| B$ means that A indirectly decreases B . Like for $A \rightarrow B$, this process involves hidden components with varying activities.

Increasing Nested Relationships

BEL also allows object of a relationship to be another statement.

- $A \Rightarrow (B \Rightarrow C)$ means that A increases the process by which B increases C . The example in the BEL Spec $p(\text{HGNC:GATA1}) \Rightarrow (\text{act}(p(\text{HGNC:ZBTB16})) \Rightarrow r(\text{HGNC:MPL}))$ represents GATA1 directly increasing the process by which ZBTB16 directly increases MPL. Before, directly increasing was used to specify physical contact, so it’s reasonable to conclude that $p(\text{HGNC:GATA1}) \Rightarrow \text{act}(p(\text{HGNC:ZBTB16}))$. The specification cites examples when B is an activity that only is affected in the context of A and C . This complicated enough that it is both impractical to standardize during curation, and impractical to represent in a network.
- $A \rightarrow (B \Rightarrow C)$ can be interpreted by assuming that A indirectly increases B , and because of monotonicity, conclude that $A \rightarrow C$ as well.
- $A \Rightarrow (B \rightarrow C)$ is more difficult to interpret, because it does not describe which part of process $B \rightarrow C$ is affected by A or how. Is it that $A \Rightarrow B$, and $B \Rightarrow C$, so we conclude $A \rightarrow C$, or does it mean something else? Perhaps A impacts a different portion of the hidden process in $B \rightarrow C$. These statements are ambiguous enough that they should be written as just $A \Rightarrow B$, and $B \rightarrow C$. If there is no literature evidence for the statement $A \rightarrow C$, then it is not the job of the curator to make this inference. Identifying statements of this might be the goal of a bioinformatics analysis of the BEL network after compilation.
- $A \rightarrow (B \rightarrow C)$ introduces even more ambiguity, and it should not be used.
- $A \Rightarrow (B =| C)$ states A increases the process by which B decreases C . One interpretation of this statement might be that $A \Rightarrow B$ and $B =| C$. An analysis could infer $A -| C$. Statements in the form of $A \rightarrow (B =| C)$ can also be resolved this way, but with added ambiguity.

Decreasing Nested Relationships

While we could agree on usage for the previous examples, the decrease of a nested statement introduces an unreasonable amount of ambiguity.

- $A = | (B \Rightarrow C)$ could mean A decreases B , and B also increases C . Does this mean A decreases C , or does it mean that C is still increased, but just not as much? Which of these statements takes precedence? Or do their effects cancel? The same can be said about $A - | (B \Rightarrow C)$, and with added ambiguity for indirect increases $A - | (B -> C)$
- $A = | (B = | C)$ could mean that A decreases B and B decreases C . We could conclude that A increases C , or could we again run into the problem of not knowing the precedence? The same is true for the indirect versions.

Recommendations for Use in PyBEL

After considering the ambiguity of nested statements to be a great risk to clarity, and PyBEL disables the usage of nested statements by default. See the Input and Output section for different parser settings. At Fraunhofer SCAI, curators resolved these statements to single statements to improve the precision and readability of our BEL documents.

While most statements in the form $A \text{ rel1 } (B \text{ rel2 } C)$ can be reasonably expanded to $A \text{ rel1 } B$ and $B \text{ rel2 } C$, the few that cannot are the difficult-to-interpret cases that we need to be careful about in our curation and later analyses.

1.5.5 Why Not RDF?

Current bel2rdf serialization tools build URLs with the OpenBEL Framework domain as a namespace, rather than respect the original namespaces of original entities. This does not follow the best practices of the semantic web, where URL's representing an object point to a real page with additional information. For example, UniProt does an exemplary job of this. Ultimately, using non-standard URLs makes harmonizing and data integration difficult.

Additionally, the RDF format does not easily allow for the annotation of edges. A simple statement in BEL that one protein up-regulates another can be easily represented in a triple in RDF, but when the annotations and citation from the BEL document need to be included, this forces RDF serialization to use approaches like representing the statement itself as a node. RDF was not intended to represent this type of information, but more properly for locating resources (hence its name). Furthermore, many blank nodes are introduced throughout the process. This makes RDF incredibly difficult to understand or work with. Later, writing queries in SPARQL becomes very difficult because the data format is complicated and the language is limited. For example, it would be incredibly complicated to write a query in SPARQL to get the objects of statements from publications by a certain author.

INSTALLATION

The latest stable code can be installed from [PyPI](#) with:

```
$ python3 -m pip install pybel
```

The most recent code can be installed from the source on [GitHub](#) with:

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

For developers, the repository can be cloned from [GitHub](#) and installed in editable mode with:

```
$ git clone https://github.com/pybel/pybel.git
$ cd pybel
$ python3 -m pip install -e .
```

2.1 Extras

The `setup.py` makes use of the `extras_require` argument of `setuptools.setup()` in order to make some heavy packages that support special features of PyBEL optional to install, in order to make the installation more lean by default. A single extra can be installed from PyPI like `python3 -m pip install pybel[neo4j]` or multiple can be installed using a list like `python3 -m pip install pybel[neo4j,indra]`. Likewise, for developer installation, extras can be installed in editable mode with `python3 -m pip install -e .[neo4j]` or multiple can be installed using a list like `python3 -m pip install -e .[neo4j,indra]`. The available extras are:

2.1.1 neo4j

This extension installs the `py2neo` package to support upload and download to Neo4j databases.

See also:

- `pybel.to_neo4j()`

2.1.2 indra

This extra installs support for `indra`, the integrated network dynamical reasoner and assembler. Because it also represents biology in BEL-like statements, many statements from PyBEL can be converted to INDRA, and visa-versa. This package also enables the import of BioPAX, SBML, and SBGN into BEL.

See also:

- `pybel.from_biopax()`
- `pybel.from_indra_statements()`
- `pybel.from_indra_pickle()`
- `pybel.to_indra()`

2.1.3 jupyter

This extra installs support for visualizing BEL graphs in Jupyter notebooks.

See also:

- `pybel.io.jupyter.to_html()`
- `pybel.io.jupyter.to_jupyter()`

2.2 Caveats

- PyBEL extends the `networkx` for its core data structure. Many of the graphical aspects of `networkx` depend on `matplotlib`, which is an optional dependency.
- If `HTMLlib5` is installed, the test that's supposed to fail on a web page being missing actually tries to parse it as RDFa, and doesn't fail. Disregard this.

2.3 Upgrading

During the current development cycle, programmatic access to the definition and graph caches might become unstable. If you have any problems working with the database, try removing it with one of the following commands:

1. Running `pybel manage drop` (unix)
2. Running `python3 -m pybel manage drop` (windows)
3. Removing the folder `~/pybel`

PyBEL will build a new database and populate it on the next run.

DATA MODEL

The `pybel.struct` module houses functions for handling the main data structure in PyBEL.

Because BEL expresses how biological entities interact within many different contexts, with descriptive annotations, PyBEL represents data as a directed multi-graph by sub-classing the `networkx.MultiDiGraph`. Each node is an instance of a subclass of the `pybel.dsl.BaseEntity` and each edge has a stable key and associated data dictionary for storing relevant contextual information.

The graph contains metadata for the PyBEL version, the BEL script metadata, the namespace definitions, the annotation definitions, and the warnings produced in analysis. Like any `networkx` graph, all attributes of a given object can be accessed through the `graph` property, like in: `my_graph.graph['my key']`. Convenient property definitions are given for these attributes that are outlined in the documentation for `pybel.BELGraph`.

This allows for much easier programmatic access to answer more complicated questions, which can be written with python code. Because the data structure is the same in Neo4J, the data can be directly exported with `pybel.to_neo4j()`. Neo4J supports the Cypher querying language so that the same queries can be written in an elegant and simple way.

3.1 Constants

These documents refer to many aspects of the data model using constants, which can be found in the top-level module `pybel.constants`.

Terms describing abundances, annotations, and other internal data are designated in `pybel.constants` with full-caps, such as `pybel.constants.FUNCTION` and `pybel.constants.PROTEIN`.

For normal usage, we suggest referring to values in dictionaries by these constants, in case the hard-coded strings behind these constants change.

3.1.1 Function Nomenclature

The following table shows PyBEL's internal mapping from BEL functions to its own constants. This can be accessed programmatically via `pybel.parser.language.abundance_labels`.

BEL Function	PyBEL Constant	PyBEL DSL
<code>a()</code> , <code>abundance()</code>	<code>pybel.constants.ABUNDANCE</code>	<code>pybel.dsl.Abundance</code>
<code>g()</code> , <code>geneAbundance()</code>	<code>pybel.constants.GENE</code>	<code>pybel.dsl.Gene</code>
<code>r()</code> , <code>rnaAbundance()</code>	<code>pybel.constants.RNA</code>	<code>pybel.dsl.Rna</code>
<code>m()</code> , <code>microRNAAbundance()</code>	<code>pybel.constants.MIRNA</code>	<code>pybel.dsl.MicroRna</code>
<code>p()</code> , <code>proteinAbundance()</code>	<code>pybel.constants.PROTEIN</code>	<code>pybel.dsl.Protein</code>
<code>bp()</code> , <code>biologicalProcess()</code>	<code>pybel.constants.BIOPROCESS</code>	<code>pybel.dsl.BiologicalProcess</code>
<code>path()</code> , <code>pathology()</code>	<code>pybel.constants.PATHOLOGY</code>	<code>pybel.dsl.Pathology</code>
<code>complex()</code> , <code>complexAbundance()</code>	<code>pybel.constants.COMPLEX</code>	<code>pybel.dsl.ComplexAbundance</code>
<code>composite()</code> , <code>compositeAbundance()</code>	<code>pybel.constants.COMPOSITE</code>	<code>pybel.dsl.CompositeAbundance</code>
<code>rxn()</code> , <code>reaction()</code>	<code>pybel.constants.REACTION</code>	<code>pybel.dsl.Reaction</code>

3.2 Graph

class `pybel.BELGraph`(*name=None, version=None, description=None, authors=None, contact=None, license=None, copyright=None, disclaimer=None, path=None*)

An extension to `networkx.MultiDiGraph` to represent BEL.

Initialize a BEL graph with its associated metadata.

Parameters

- **name** (`Optional[str]`) – The graph’s name
- **version** (`Optional[str]`) – The graph’s version. Recommended to use semantic versioning or YYYYMMDD format.
- **description** (`Optional[str]`) – A description of the graph
- **authors** (`Optional[str]`) – The authors of this graph
- **contact** (`Optional[str]`) – The contact email for this graph
- **license** (`Optional[str]`) – The license for this graph
- **copyright** (`Optional[str]`) – The copyright for this graph
- **disclaimer** (`Optional[str]`) – The disclaimer for this graph

__add__(*other*)

Copy this graph and join it with another graph with it using `pybel.struct.left_full_join()`.

Parameters *other* (`BELGraph`) – Another BEL graph

Example usage:

```
>>> from pybel.examples import ras_tloc_graph, braf_graph
>>> k = ras_tloc_graph + braf_graph
```

Return type `BELGraph`

__iadd__(*other*)

Join another graph into this one, in-place, using `pybel.struct.left_full_join()`.

Parameters *other* (*BELGraph*) – Another BEL graph

Example usage:

```
>>> from pybel.examples import ras_tloc_graph, braf_graph
>>> ras_tloc_graph += braf_graph
```

Return type *BELGraph*

__and__ (*other*)

Create a deep copy of this graph and left outer joins another graph.

Uses *pybel.struct.left_outer_join()*.

Parameters *other* (*BELGraph*) – Another BEL graph

Example usage:

```
>>> from pybel.examples import ras_tloc_graph, braf_graph
>>> k = ras_tloc_graph & braf_graph
```

Return type *BELGraph*

__iand__ (*other*)

Join another graph into this one, in-place, using *pybel.struct.left_outer_join()*.

Parameters *other* (*BELGraph*) – Another BEL graph

Example usage:

```
>>> from pybel.examples import ras_tloc_graph, braf_graph
>>> ras_tloc_graph &= braf_graph
```

Return type *BELGraph*

transitivities: *Set[Tuple[str, str]]*

A set of pairs of hashes of edges over which there is transitivity. For example, for the nested statement (P(X) -> P(Y)) -> P(Z) will have a pair for (hash(P(X) -> P(Y)), hash(P(Y) -> P(Z)))

parent

A reference to the parent graph

child()

Create an empty graph with a “parent” reference back to this one.

Return type *BELGraph*

property count: *pybel.struct.graph.CountDispatch*

A dispatch to count functions.

Can be used like this:

```
>>> from pybel.examples import sialic_acid_graph
>>> sialic_acid_graph.count.functions()
Counter({'Protein': 7, 'Complex': 1, 'Abundance': 1})
```

Return type *CountDispatch*

property summarize: `pybel.struct.graph.SummarizeDispatch`

A dispatch to summarize the graph.

Return type *SummarizeDispatch*

property expand: `pybel.struct.graph.ExpandDispatch`

A dispatch to expand the graph w.r.t. its parent.

Return type *ExpandDispatch*

property induce: `pybel.struct.graph.InduceDispatch`

A dispatch to mutate the graph.

Return type *InduceDispatch*

property plot: `pybel.struct.graph.PlotDispatch`

A dispatch to plot the graph using matplotlib and seaborn.

Return type *PlotDispatch*

property path: `Optional[str]`

The graph's path, if it was derived from a BEL document.

Return type `Optional[str]`

property name: `Optional[str]`

The graph's name.

Hint: Can be set with the SET DOCUMENT Name = "... " entry in the source BEL script.

Return type `Optional[str]`

property version: `Optional[str]`

The graph's version.

Hint: Can be set with the SET DOCUMENT Version = "... " entry in the source BEL script.

Return type `Optional[str]`

property description: `Optional[str]`

The graph's description.

Hint: Can be set with the SET DOCUMENT Description = "... " entry in the source BEL document.

Return type `Optional[str]`

property authors: `Optional[str]`

The graph's authors.

Hint: Can be set with the SET DOCUMENT Authors = "... " entry in the source BEL document.

Return type `Optional[str]`

property contact: `Optional[str]`

The graph's contact information.

Hint: Can be set with the SET DOCUMENT `ContactInfo = "..."` entry in the source BEL document.

Return type `Optional[str]`

property license: `Optional[str]`

The graph's license.

Hint: Can be set with the SET DOCUMENT `Licenses = "..."` entry in the source BEL document

Return type `Optional[str]`

property copyright: `Optional[str]`

The graph's copyright.

Hint: Can be set with the SET DOCUMENT `Copyright = "..."` entry in the source BEL document

Return type `Optional[str]`

property disclaimer: `Optional[str]`

The graph's disclaimer.

Hint: Can be set with the SET DOCUMENT `Disclaimer = "..."` entry in the source BEL document.

Return type `Optional[str]`

property namespace_url: `Dict[str, str]`

The mapping from the keywords used in this graph to their respective BEL namespace URLs.

Hint: Can be appended with the DEFINE NAMESPACE `[key] AS URL "[value]"` entries in the definitions section of the source BEL document.

Return type `Dict[str, str]`

property defined_namespace_keywords: `Set[str]`

The set of all keywords defined as namespaces in this graph.

Return type `Set[str]`

property namespace_pattern: `Dict[str, str]`

The mapping from the namespace keywords used to create this graph to their regex patterns.

Hint: Can be appended with the DEFINE NAMESPACE `[key] AS PATTERN "[value]"` entries in the definitions section of the source BEL document.

Return type `Dict[str, str]`

property annotation_url: `Dict[str, str]`

The mapping from the annotation keywords used to create this graph to the URLs of the BELANNO files.

Hint: Can be appended with the `DEFINE ANNOTATION [key] AS URL "[value]"` entries in the definitions section of the source BEL document.

Return type `Dict[str, str]`

property annotation_miriam: `Set[str]`

The set of annotations defined by MIRIAM.

Return type `Set[str]`

property annotation_curie: `Set[str]`

The set of annotations defined by CURIE.

Return type `Set[str]`

property annotation_pattern: `Dict[str, str]`

The mapping from the annotation keywords used to create this graph to their regex patterns as strings.

Hint: Can be appended with the `DEFINE ANNOTATION [key] AS PATTERN "[value]"` entries in the definitions section of the source BEL document.

Return type `Dict[str, str]`

property annotation_list: `Dict[str, Set[str]]`

The mapping from the keywords of locally defined annotations to their respective sets of values.

Hint: Can be appended with the `DEFINE ANNOTATION [key] AS LIST {"[value]", ...}` entries in the definitions section of the source BEL document.

Return type `Dict[str, Set[str]]`

property defined_annotation_keywords: `Set[str]`

Get the set of all keywords defined as annotations in this graph.

Return type `Set[str]`

property pybel_version: `str`

The version of PyBEL with which this graph was produced as a string.

Return type `str`

property warnings: `List[Tuple[Optional[str], pybel.exceptions.BELParserWarning, Mapping]]`

A list of warnings associated with this graph.

Return type `List[Tuple[Optional[str], BELParserWarning, Mapping]]`

number_of_warnings()

Return the number of warnings.

Return type `int`

number_of_citations()

Return the number of citations contained within the graph.

Return type `int`

number_of_authors()

Return the number of authors contained within the graph.

Return type `int`

get_authors()

Get the authors for the citations in the graph.

Return type `Set[str]`

add_transitivity(*k1*, *k2*)

Add a pair of edge hashes over which there is transitivity.

Parameters

- **k1** (`str`) – The hash of the subject edge
- **k2** (`str`) – The hash of the object edge

Return type `None`

add_unqualified_edge(*source*, *target*, *relation*)

Add a unique edge that has no annotations.

Parameters

- **source** (`BaseEntity`) – The source node
- **target** (`BaseEntity`) – The target node
- **relation** (`str`) – A relationship label from `pybel.constants`

Return type `str`

Returns The key for this edge (a unique hash)

add_transcription(*gene*, *rna*)

Add a transcription relation from a gene to an RNA or miRNA node.

Parameters

- **gene** (`Gene`) – A gene node
- **rna** (`Union[Rna, MicroRna]`) – An RNA or microRNA node

Return type `str`

add_translation(*rna*, *protein*)

Add a translation relation from a RNA to a protein.

Parameters

- **rna** (`Rna`) – An RNA node
- **protein** (`Protein`) – A protein node

Return type `str`

**add_equivalence(*source*: `pybel.dsl.node_classes.BaseEntity`, *target*: `pybel.dsl.node_classes.BaseEntity`,
args*, *kwargs*) → `str`**

Add two equivalence relations for the nodes.

add_orthology(*source*: pybel.dsl.node_classes.BaseEntity, *target*: pybel.dsl.node_classes.BaseEntity, *args, **kwargs) → str

Add two orthology relations for the nodes such that *u* orthologousTo *v* and *v* orthologousTo *u*.

add_is_a(*source*: pybel.dsl.node_classes.BaseEntity, *target*: pybel.dsl.node_classes.BaseEntity, *, *relation*: str = 'isA') → str

Add an isA relationship such that *u* isA *v*.

add_part_of(*source*: pybel.dsl.node_classes.BaseEntity, *target*: pybel.dsl.node_classes.BaseEntity, *, *relation*: str = 'partOf') → str

Add a partOf relationship such that *u* partOf *v*.

add_has_variant(*source*: pybel.dsl.node_classes.BaseEntity, *target*: pybel.dsl.node_classes.BaseEntity, *, *relation*: str = 'hasVariant') → str

Add a hasVariant relationship such that *u* hasVariant *v*.

add_has_reactant(*source*: pybel.dsl.node_classes.BaseEntity, *target*: pybel.dsl.node_classes.BaseEntity, *, *relation*: str = 'hasReactant') → str

Add a hasReactant relationship such that *u* hasReactant *v*.

add_has_product(*source*: pybel.dsl.node_classes.BaseEntity, *target*: pybel.dsl.node_classes.BaseEntity, *, *relation*: str = 'hasProduct') → str

Add a hasProduct relationship such that *u* hasProduct *v*.

add_qualified_edge(*source*, *target*, *, *relation*, *evidence*, *citation*, *annotations*=None, *source_modifier*=None, *target_modifier*=None, **attr)

Add a qualified edge.

Qualified edges have a relation, evidence, citation, and optional annotations, subject modifications, and object modifications.

Parameters

- **source** (*BaseEntity*) – The source node
- **target** (*BaseEntity*) – The target node
- **relation** (str) – The type of relation this edge represents
- **evidence** (str) – The evidence string from an article
- **citation** (Union[str, Tuple[str, str], CitationDict]) – The citation data dictionary for this evidence. If a string is given, assumes it's a PubMed identifier and auto-fills the citation type.
- **annotations** (Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[Entity]], None]) – The annotations data dictionary
- **source_modifier** (Optional[Mapping[str, Any]]) – The modifiers (like activity) on the subject node. See data model documentation.
- **target_modifier** (Optional[Mapping[str, Any]]) – The modifiers (like activity) on the object node. See data model documentation.

Return type str

Returns The hash of the edge

add_binds(*source*, *target*, *, *evidence*, *citation*, *annotations*=None, **attr)

Add a “binding” relationship between the two entities such that *u* => complex(*u*, *v*).

Return type str

add_increases(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'increases', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Wrap `add_qualified_edge()` for the `pybel.constants.INCREASES` relation.

add_directly_increases(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'directlyIncreases', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Add a `pybel.constants.DIRECTLY_INCREASES` with `add_qualified_edge()`.

add_decreases(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'decreases', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Add a `pybel.constants.DECREASES` relationship with `add_qualified_edge()`.

add_directly_decreases(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'directlyDecreases', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Add a `pybel.constants.DIRECTLY_DECREASES` relationship with `add_qualified_edge()`.

add_association(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *args, **kwargs) → str

Add a `pybel.constants.ASSOCIATION` relationship with `add_qualified_edge()`.

add_regulates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'regulates', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Add a `pybel.constants.REGULATES` relationship with `add_qualified_edge()`.

add_directly_regulates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'directlyRegulates', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Add a `pybel.constants.DIRECTLY_REGULATES` relationship with `add_qualified_edge()`.

add_correlation(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *args, **kwargs) → str

Add a `pybel.constants.CORRELATION` relationship with `add_qualified_edge()`.

add_no_correlation(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *args, **kwargs) → str

Add a `pybel.constants.NO_CORRELATION` relationship with `add_qualified_edge()`.

add_positive_correlation(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *args, **kwargs) → str

Add a `pybel.constants.POSITIVE_CORRELATION` relationship with `add_qualified_edge()`.

add_negative_correlation(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *args, **kwargs) → str

Add a `pybel.constants.NEGATIVE_CORRELATION` relationship with `add_qualified_edge()`.

add_causes_no_change(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'causesNoChange', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr) → str

Add a `pybel.constants.CAUSES_NO_CHANGE` relationship with `add_qualified_edge()`.

add_inhibits(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'decreases', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, {'effect': {'identifier': '0003674', 'name': 'molecular function', 'namespace': 'go'}, 'modifier': 'Activity'}, **attr) → str

Add an “inhibits” relationship.

A more specific version of `add_decreases()` that automatically populates the object modifier with an activity.

add_activates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.BaseEntity, *, relation: str = 'increases', evidence: str, citation: Union[str, Tuple[str, str], pybel.language.CitationDict], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, {'effect': {'identifier': '0003674', 'name': 'molecular function', 'namespace': 'go'}, 'modifier': 'Activity'}, **attr) → str

Add an “activates” relationship.

A more specific version of `add_increases()` that automatically populates the object modifier with an activity.

add_phosphorylates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.CentralDogma, code: Optional[str] = None, position: Optional[int] = None, *, evidence: str, citation: Union[str, Mapping[str, str]], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping[str, Any]] = None, target_modifier: Optional[Mapping[str, Any]] = None, **attr)

Add an increase of modified object with phosphorylation.

add_directly_phosphorylates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.CentralDogma, code: Optional[str] = None, position: Optional[int] = None, *, evidence: str, citation: Union[str, Mapping[str, str]], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping] = None, target_modifier: Optional[Mapping] = None, **attr)

Add a direct increase of modified object with phosphorylation.

add_dephosphorylates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.CentralDogma, code: Optional[str] = None, position: Optional[int] = None, *, evidence: str, citation: Union[str, Mapping[str, str]], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping] = None, target_modifier: Optional[Mapping] = None, **attr)

Add a decrease of modified object with phosphorylation.

add_directly_dephosphorylates(source: pybel.dsl.node_classes.BaseEntity, target: pybel.dsl.node_classes.CentralDogma, code: Optional[str] = None, position: Optional[int] = None, *, evidence: str, citation: Union[str, Mapping[str, str]], annotations: Optional[Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[pybel.language.Entity]]]] = None, source_modifier: Optional[Mapping] = None, target_modifier: Optional[Mapping] = None, **attr)

Add a direct decrease of modified object with phosphorylation.

add_node_from_data(node)

Add an entity to the graph.

Return type None

add_reaction(reactants, products)

Add a reaction directly to the graph.

Return type None

has_edge_citation(u, v, key)

Check if the given edge has a citation.

Return type bool

has_edge_evidence(u, v, key)

Check if the given edge has an evidence.

Return type bool

get_edge_citation(u, v, key)

Get the citation for a given edge.

Return type Optional[CitationDict]

get_edge_evidence(u, v, key)

Get the evidence for a given edge.

Return type Optional[str]

get_edge_annotations(u, v, key)

Get the annotations for a given edge.

Return type Optional[Mapping[str, List[Entity]]]

static `node_to_bel(n)`
Serialize a node as BEL.

Return type `str`

static `edge_to_bel(u, v, edge_data, sep=None, use_identifiers=True)`
Serialize a pair of nodes and related edge data as a BEL relation.

Return type `str`

iter_equivalent_nodes(*node*)
Iterate over nodes that are equivalent to the given node, including the original.

Return type `Iterable[BaseEntity]`

get_equivalent_nodes(*node*)
Get a set of equivalent nodes to this node, excluding the given node.

Return type `Set[BaseEntity]`

node_has_namespace(*node*, *namespace*)
Check if the node have the given namespace.

This also should look in the equivalent nodes.

Return type `bool`

summary_dict()
Return a dictionary that summarizes the graph.

Return type `Mapping[str, float]`

summary_str()
Return a string that summarizes the graph.

Return type `str`

ground(*kwargs*)**
Ground this graph.

Return type `BELGraph`

3.2.1 Dispatches

Dispatches are classes that enable easy access to summary, mutation, and other functions that consume graphs directly through the `pybel.BELGraph` interface.

class `pybel.struct.graph.CountDispatch(graph)`
A dispatch for count functions that can be found at `pybel.BELGraph.count`.

functions()
Count the functions in a graph.

```
>>> from pybel.examples import sialic_acid_graph
>>> sialic_acid_graph.count.functions()
Counter({'Protein': 7, 'Complex': 1, 'Abundance': 1})
```

Return type `Counter`

namespaces()
Return a counter of namespaces' occurrences in nodes in the graph.

Return type `Counter`

pathologies()

Return a counter of pathologies' occurrences in edges in the graph.

Return type `Counter`

annotations()

Return a counter of annotations' occurrences in edges in the graph.

Return type `Counter`

variants()

Return a counter of variants' occurrences in nodes in the graph.

Return type `Counter`

relations()

Return a counter of relations' occurrences in edges in the graph.

Return type `Counter`

error_types()

Return a counter of error types' occurrences in BEL script underlying the graph.

Return type `Counter`

modifications()

Return a counter of relation modifications' occurrences (activity, translocation, etc.) in the graph.

Return type `Counter`

authors()

Return a counter of the number of edges to which each author contributed in the graph.

Return type `Counter`

citations()

Return the number of citations.

Return type `int`

class `pybel.struct.graph.InduceDispatch(graph)`

A dispatch for induction functions that can be found at [`pybel.BELGraph.induce`](#).

paths(nodes)

Induce a subgraph on shortest paths between the nodes.

Return type `Optional[BELGraph]`

neighborhood(nodes)

Induce a subgraph around the neighborhood.

Return type `Optional[BELGraph]`

random(kwargs)**

Induce a random subgraph.

Return type `Optional[BELGraph]`

annotation(prefix, identifier)

Induce a subgraph on edges with the given annotation.

Return type `Optional[BELGraph]`

class `pybel.struct.graph.SummarizeDispatch(graph)`

A dispatch for summary printing functions that can be found at [`pybel.BELGraph.summarize`](#).

statistics(*file=None*)

Print summary statistics on the graph.

nodes(*file=None, examples=True*)

Print a summary of the nodes' functions in the graph.

namespaces(*file=None, examples=True*)

Print a summary of the nodes' namespaces in the graph.

edges(*file=None, examples=True*)

Print a summary of the edges' types in the graph.

citations(*n=15, file=None*)

Print a summary of the top citations' frequencies in the graph.

dict()

Return a dictionary that summarizes the graph.

Return type `Mapping[str, float]`

str(***kwargs*)

Return a string that summarizes the graph.

Return type `str`

list()

Return a list of tuples that summarize the graph.

Return type `List[Tuple[str, Any]]`

class `pybel.struct.graph.ExpandDispatch`(*graph*)

A dispatch for count functions that can be found at [`pybel.BELGraph.expand`](#).

property `parent`: `pybel.struct.graph.BELGraph`

Get the parent BEL graph.

Return type `BELGraph`

neighborhood(*node*)

Expand around the neighborhood of a given node.

```
>>> from pybel.examples import braf_graph
>>> from pybel.dsl import Protein
>>> thpo = Protein(namespace='HGNC', name='THPO', identifier='11795')
>>> braf = Protein(namespace='HGNC', name='BRAF', identifier='1097')
>>> raf1 = Protein(namespace='HGNC', name='RAF1', identifier='9829')
>>> elk1 = Protein(namespace='HGNC', name='ELK1', identifier='3321')
>>> subgraph_1 = braf_graph.induce.paths([braf, elk1])
>>> assert thpo not in subgraph_1 and raf1 not in subgraph_1
>>> subgraph_2 = subgraph_1.expand.neighborhood(braf)
>>> assert thpo in subgraph_2 and raf1 not in subgraph_2
```

Return type `BELGraph`

periphery(***kwargs*)

Expand around the periphery of the graph w.r.t. its parent graph.

internal(***kwargs*)

Expand missing edges between nodes in the graph w.r.t. its parent graph.

class `pybel.struct.graph.PlotDispatch`(*graph*)

A dispatch for count functions that can be found at [`pybel.BELGraph.plot`](#).

summary(*save=None, **kwargs*)

Plot a summary of the graph's nodes and edges using `matplotlib`.

3.3 Nodes

Nodes (or *entities*) in a `pybel.BELGraph` represent physical entities' abundances. Most contain information about the identifier for the entity using a namespace/name pair. The PyBEL parser converts BEL terms to an internal representation using an internal domain specific language (DSL) that allows for writing BEL directly in Python.

For example, after the BEL term `p(hgnc:GSK3B)` is parsed, it is instantiated as a Python object using the DSL function corresponding to the `p()` function in BEL, `pybel.dsl.Protein`, like:

```
from pybel.dsl import Protein
gsk3b_protein = Protein(namespace='hgnc', name='GSK3B')
```

`pybel.dsl.Protein`, like the others mentioned before, inherit from `pybel.dsl.BaseEntity`, which itself inherits from `dict`. Therefore, the resulting object can be used like a dict that looks like:

```
from pybel.constants import *

{
    FUNCTION: PROTEIN,
    NAMESPACE: 'hgnc',
    NAME: 'GSK3B',
}
```

Alternatively, it can be used in more exciting ways, outlined later in the documentation for `pybel.dsl`.

3.3.1 Variants

The addition of a variant tag results in an entry called 'variants' in the data dictionary associated with a given node. This entry is a list with dictionaries describing each of the variants. All variants have the entry 'kind' to identify whether it is a post-translational modification (PTM), gene modification, fragment, or HGVS variant.

Warning: The canonical ordering for the elements of the `VARIANTS` list correspond to the sorted order of their corresponding node tuples using `pybel.parser.canonicalize.sort_dict_list()`. Rather than directly modifying the `BELGraph`'s structure, use `pybel.BELGraph.add_node_from_data()`, which takes care of automatically canonicalizing this dictionary.

HGVS Variants.

For example, the BEL term `p(HGNC:GSK3B, var(p.Gly123Arg))` is translated to the following internal DSL:

```
from pybel.dsl import Protein, Hgvs
gsk3b_variant = Protein(namespace='HGNC', name='GSK3B', variants=Hgvs('p.Gly123Arg'))
```

Further, the shorthand for protein substitutions, `pybel.dsl.ProteinSubstitution`, can be used to produce the same result, as it inherits from `pybel.dsl.Hgvs`:

```
from pybel.dsl import Protein, ProteinSubstitution
gsk3b_variant = Protein(namespace='HGNC', name='GSK3B', variants=ProteinSubstitution('Gly
↪', 123, 'Arg'))
```


Either way, the resulting object can be used like a dict that looks like:

```
from pybel.constants import *

{
    FUNCTION: PROTEIN,
    NAMESPACE: 'HGNC',
    NAME: 'GSK3B',
    VARIANTS: [
        {
            KIND: HGVS,
            IDENTIFIER: 'p.Gly123Arg',
        },
    ],
}
```

See also:

- BEL 2.0 specification on [variants](#)
- HGVS [conventions](#)
- PyBEL module `pybel.parser.modifiers.get_hgvs_language`

3.3.2 Gene Substitutions

Gene Substitutions.

Gene substitutions are legacy statements defined in BEL 1.0. BEL 2.0 recommends using HGVS strings. Luckily, the information contained in a BEL 1.0 encoding, such as `g(HGNC:APP,sub(G,275341,C))` can be automatically translated to the appropriate HGVS `g(HGNC:APP, var(c.275341G>C))`, assuming that all substitutions are using the reference coding gene sequence for numbering and not the genomic reference. The previous statements both produce the underlying data:

```
from pybel.constants import *

{
    FUNCTION: GENE,
    NAMESPACE: 'HGNC',
    NAME: 'APP',
    VARIANTS: [
        {
            KIND: HGVS,
            IDENTIFIER: 'c.275341G>C',
        },
    ],
}
```

See also:

- BEL 2.0 specification on [gene substitutions](#)
- PyBEL module `pybel.parser.modifiers.get_gene_substitution_language`

3.3.3 Gene Modifications

Gene Modifications.

PyBEL introduces the gene modification tag, `gmod()`, to allow for the encoding of epigenetic modifications. Its syntax follows the same style as the `pmod()` tags for proteins, and can include the following values:

- M
- Me
- methylation
- A
- Ac
- acetylation

For example, the node `g(HGNC:GSK3B, gmod(M))` is represented with the following:

```
from pybel.constants import *

{
    FUNCTION: GENE,
    NAMESPACE: 'HGNC',
    NAME: 'GSK3B',
    VARIANTS: [
        {
            KIND: GMOD,
            IDENTIFIER: {
                NAMESPACE: BEL_DEFAULT_NAMESPACE,
                NAME: 'Me',
            },
        },
    ],
}
```

The addition of this function does not preclude the use of all other standard functions in BEL; however, other compilers probably won't support these standards. If you agree that this is useful, please contribute to discussion in the OpenBEL community.

See also:

- PyBEL module `pybel.parser.modifiers.get_gene_modification_language()`

3.3.4 Protein Substitutions

Protein Substitutions.

Protein substitutions are legacy statements defined in BEL 1.0. BEL 2.0 recommends using HGVS strings. Luckily, the information contained in a BEL 1.0 encoding, such as `p(HGNC:APP, sub(R, 275, H))` can be automatically translated to the appropriate HGVS `p(HGNC:APP, var(p.Arg275His))`, assuming that all substitutions are using the reference protein sequence for numbering and not the genomic reference. The previous statements both produce the underlying data:

```
from pybel.constants import *
```

(continues on next page)

(continued from previous page)

```
{
  FUNCTION: GENE,
  NAMESPACE: 'HGNC',
  NAME: 'APP',
  VARIANTS: [
    {
      KIND: HGVS,
      IDENTIFIER: 'p.Arg275His',
    },
  ],
}
```

See also:

- BEL 2.0 specification on [protein substitutions](#)
- PyBEL module `pybel.parser.modifiers.get_protein_substitution_language`

3.3.5 Protein Modifications

Protein Modifications.

The addition of a post-translational modification (PTM) tag results in an entry called ‘variants’ in the data dictionary associated with a given node. This entry is a list with dictionaries describing each of the variants. All variants have the entry ‘kind’ to identify whether it is a PTM, gene modification, fragment, or HGVS variant. The ‘kind’ value for PTM is ‘pmod’.

Each PMOD contains an identifier, which is a dictionary with the namespace and name, and can optionally include the position (‘pos’) and/or amino acid code (‘code’).

For example, the node `p(HGNC:GSK3B, pmod(P, S, 9))` is represented with the following:

```
from pybel.constants import *

{
  FUNCTION: PROTEIN,
  NAMESPACE: 'HGNC',
  NAME: 'GSK3B',
  VARIANTS: [
    {
      KIND: PMOD,
      IDENTIFIER: {
        NAMESPACE: BEL_DEFAULT_NAMESPACE
        NAME: 'Ph',
      },
      PMOD_CODE: 'Ser',
      PMOD_POSITION: 9,
    },
  ],
}
```

As an additional example, in `p(HGNC:MAPK1, pmod(Ph, Thr, 202), pmod(Ph, Tyr, 204))`, MAPK is phosphorylated twice to become active. This results in the following:

```
{
  FUNCTION: PROTEIN,
  NAMESPACE: 'HGNC',
  NAME: 'MAPK1',
  VARIANTS: [
    {
      KIND: PMOD,
      IDENTIFIER: {
        NAMESPACE: BEL_DEFAULT_NAMESPACE
        NAME: 'Ph',
      },
      PMOD_CODE: 'Thr',
      PMOD_POSITION: 202
    },
    {
      KIND: PMOD,
      IDENTIFIER: {
        NAMESPACE: BEL_DEFAULT_NAMESPACE
        NAME: 'Ph',
      },
      PMOD_CODE: 'Tyr',
      PMOD_POSITION: 204
    }
  ]
}
```

See also:

- BEL 2.0 specification on [protein modifications](#)
- PyBEL module `pybel.parser.modifiers.get_protein_modification_language`

3.3.6 Protein Truncations

Truncations.

Truncations in the legacy BEL 1.0 specification are automatically translated to BEL 2.0 with HGVS nomenclature. `p(HGNC:AKT1, trunc(40))` becomes `p(HGNC:AKT1, var(p.40*))` and is represented with the following dictionary:

```
from pybel.constants import *

{
  FUNCTION: PROTEIN,
  NAMESPACE: 'HGNC',
  NAME: 'AKT1',
  VARIANTS: [
    {
      KIND: HGVS,
      IDENTIFIER: 'p.40*',
    },
  ],
}
```

(continues on next page)

(continued from previous page)

```
    ],
}
```

Unfortunately, the HGVS nomenclature requires the encoding of the terminal amino acid which is exchanged for a stop codon, and this information is not required by BEL 1.0. For this example, the proper encoding of the truncation at position also includes the information that the 40th amino acid in the AKT1 is Cys. Its BEL encoding should be `p(HGNC:AKT1, var(p.Cys40*))`. Temporary support has been added to compile these statements, but it's recommended they are upgraded by reexamining the supporting text, or looking up the amino acid sequence.

See also:

- BEL 2.0 specification on [truncations](#)
- PyBEL module `pybel.parser.modifiers.get_truncation_language`

3.3.7 Protein Fragments

Fragments.

The addition of a fragment results in an entry called `pybel.constants.VARIANTS` in the data dictionary associated with a given node. This entry is a list with dictionaries describing each of the variants. All variants have the entry `pybel.constants.KIND` to identify whether it is a PTM, gene modification, fragment, or HGVS variant. The `pybel.constants.KIND` value for a fragment is `pybel.constants.FRAGMENT`.

Each fragment contains an identifier, which is a dictionary with the namespace and name, and can optionally include the position ('pos') and/or amino acid code ('code').

For example, the node `p(HGNC:GSK3B, frag(45_129))` is represented with the following:

```
from pybel.constants import *

{
    FUNCTION: PROTEIN,
    NAMESPACE: 'HGNC',
    NAME: 'GSK3B',
    VARIANTS: [
        {
            KIND: FRAGMENT,
            FRAGMENT_START: 45,
            FRAGMENT_STOP: 129,
        },
    ],
}
```

Additionally, nodes can have an asterick (*) or question mark (?) representing unbound or unknown fragments, respectively.

A fragment may also be unknown, such as in the node `p(HGNC:GSK3B, frag(?))`. This is represented with the key `pybel.constants.FRAGMENT_MISSING` and the value of '?' like:

```
from pybel.constants import *

{
    FUNCTION: PROTEIN,
    NAMESPACE: 'HGNC',
```

(continues on next page)

(continued from previous page)

```

NAME: 'GSK3B',
VARIANTS: [
    {
        KIND: FRAGMENT,
        FRAGMENT_MISSING: '?',
    },
],
}

```

See also:

- BEL 2.0 specification on [proteolytic fragments \(2.2.3\)](#)
- PyBEL module `pybel.parser.modifiers.get_fragment_language`

3.3.8 Fusions

Fusions.

Gene, RNA, miRNA, and protein fusions are all represented with the same underlying data structure. Below it is shown with uppercase letters referring to constants from `pybel.constants` and. For example, `g(HGNC:BCR, fus(HGNC:JAK2, 1875, 2626))` is represented as:

```

from pybel.constants import *

{
    FUNCTION: GENE,
    FUSION: {
        PARTNER_5P: {NAMESPACE: 'HGNC', NAME: 'BCR'},
        PARTNER_3P: {NAMESPACE: 'HGNC', NAME: 'JAK2'},
        RANGE_5P: {
            FUSION_REFERENCE: 'c',
            FUSION_START: '?',
            FUSION_STOP: 1875,
        },
        RANGE_3P: {
            FUSION_REFERENCE: 'c',
            FUSION_START: 2626,
            FUSION_STOP: '?',
        },
    },
}

```

See also:

- BEL 2.0 specification on [fusions \(2.6.1\)](#)
- PyBEL module `pybel.parser.modifiers.get_fusion_language`
- PyBEL module `pybel.parser.modifiers.get_legacy_fusion_language`

3.4 Unqualified Edges

Unqualified edges are automatically inferred by PyBEL and do not contain citations or supporting evidence.

3.4.1 Variant and Modifications' Parent Relations

All variants, modifications, fragments, and truncations are connected to their parent entity with an edge having the relationship `hasParent`.

For `p(hgnc:GSK3B, var(p.Gly123Arg))`, the following edge is inferred:

```
p(hgnc:GSK3B, var(p.Gly123Arg)) hasParent p(hgnc:GSK3B)
```

All variants have this relationship to their reference node. BEL does not specify relationships between variants, such as the case when a given phosphorylation is necessary to make another one. This knowledge could be encoded directly like BEL, since PyBEL does not restrict users from manually asserting unqualified edges.

3.4.2 List Abundances

Complexes and composites that are defined by lists. As of version 0.9.0, they contain a list of the data dictionaries that describe their members. For example `complex(p(hgnc:FOS), p(hgnc:JUN))` becomes:

```
from pybel.constants import *

{
    FUNCTION: COMPLEX,
    MEMBERS: [
        {
            FUNCTION: PROTEIN,
            NAMESPACE: 'hgnc',
            NAME: 'FOS',
        }, {
            FUNCTION: PROTEIN,
            NAMESPACE: 'hgnc',
            NAME: 'JUN',
        }
    ]
}
```

The following edges are also inferred:

```
complex(p(hgnc:FOS), p(hgnc:JUN)) hasMember p(hgnc:FOS)
complex(p(hgnc:FOS), p(hgnc:JUN)) hasMember p(hgnc:JUN)
```

See also:

BEL 2.0+ Tutorial on [complex abundances](#)

Similarly, `composite(a(CHEBI:malonate), p(hgnc:JUN))` becomes:

```
from pybel.constants import *

{
```

(continues on next page)

(continued from previous page)

```

FUNCTION: COMPOSITE,
MEMBERS: [
    {
        FUNCTION: ABUNDANCE,
        NAMESPACE: 'CHEBI',
        NAME: 'malonate',
    }, {
        FUNCTION: PROTEIN,
        NAMESPACE: 'hgnc',
        NAME: 'JUN',
    }
]
}

```

The following edges are inferred:

```

composite(a(CHEBI:malonate), p(hgnc:JUN)) hasComponent a(CHEBI:malonate)
composite(a(CHEBI:malonate), p(hgnc:JUN)) hasComponent p(hgnc:JUN)

```

Warning: The canonical ordering for the elements of the `pybel.constants.MEMBERS` list correspond to the sorted order of their corresponding node tuples using `pybel.parser.canonicalize.sort_dict_list()`. Rather than directly modifying the `BELGraph`'s structure, use `BELGraph.add_node_from_data()`, which takes care of automatically canonicalizing this dictionary.

See also:

BEL 2.0+ Tutorial on [composite abundances](#)

3.4.3 Reactions

The usage of a reaction causes many nodes and edges to be created. The following example will illustrate what is added to the network for

```

rxn(reactants(a(CHEBI:"(3S)-3-hydroxy-3-methylglutaryl-CoA"), a(CHEBI:"NADPH"), \
    a(CHEBI:"hydron")), products(a(CHEBI:"mevalonate"), a(CHEBI:"NADP(+)")))

```

As of version 0.9.0, the reactants' and products' data dictionaries are included as sub-lists keyed `REACTANTS` and `PRODUCTS`. It becomes:

```

from pybel.constants import *

{
    FUNCTION: REACTION
    REACTANTS: [
        {
            FUNCTION: ABUNDANCE,
            NAMESPACE: 'CHEBI',
            NAME: '(3S)-3-hydroxy-3-methylglutaryl-CoA'
        }, {
            FUNCTION: ABUNDANCE,

```

(continues on next page)

(continued from previous page)

```

        NAMESPACE: 'CHEBI',
        NAME: 'NADPH',
    }, {
        FUNCTION: ABUNDANCE,
        NAMESPACE: 'CHEBI',
        NAME: 'hydron',
    }
],
PRODUCTS: [
    {
        FUNCTION: ABUNDANCE,
        NAMESPACE: 'CHEBI',
        NAME: 'mevalonate',
    }, {
        FUNCTION: ABUNDANCE,
        NAMESPACE: 'CHEBI',
        NAME: 'NADP(+)',
    }
]
}

```

Warning: The canonical ordering for the elements of the REACTANTS and PRODUCTS lists correspond to the sorted order of their corresponding node tuples using `pybel.parser.canonicalize.sort_dict_list()`. Rather than directly modifying the BELGraph's structure, use `BELGraph.add_node_from_data()`, which takes care of automatically canonicalizing this dictionary.

The following edges are inferred, where X represents the previous reaction, for brevity:

```

X hasReactant a(CHEBI:"(3S)-3-hydroxy-3-methylglutaryl-CoA")
X hasReactant a(CHEBI:"NADPH")
X hasReactant a(CHEBI:"hydron")
X hasProduct a(CHEBI:"mevalonate")
X hasProduct a(CHEBI:"NADP(+)")

```

See also:

BEL 2.0+ tutorial on [reactions](#)

3.5 Edges

3.5.1 Design Choices

In the OpenBEL Framework, modifiers such as activities (`kinaseActivity`, etc.) and transformations (translocations, degradations, etc.) were represented as their own nodes. In PyBEL, these modifiers are represented as a property of the edge. In reality, an edge like `sec(p(hgnc:A)) -> activity(p(hgnc:B), ma(kinaseActivity))` represents a connection between `hgnc:A` and `hgnc:B`. Each of these modifiers explains the context of the relationship between these physical entities. Further, querying a network where these modifiers are part of a relationship is much more straightforward. For example, finding all proteins that are upregulated by the kinase activity of another protein now can be directly queried by filtering all edges for those with a subject modifier whose modification is molecular activity, and whose effect is kinase activity. Having fewer nodes also allows for a much easier display and visual interpretation

of a network. The information about the modifier on the subject and activity can be displayed as a color coded source and terminus of the connecting edge.

The compiler in OpenBEL framework created nodes for molecular activities like `kin(p(hgnc:YFG))` and induced an edge like `p(hgnc:YFG) actsIn kin(p(hgnc:YFG))`. For transformations, a statement like `tloc(p(hgnc:YFG), GO:intracellular, GO:"cell membrane")` also induced `tloc(p(hgnc:YFG), GO:intracellular, GO:"cell membrane") translocates p(hgnc:YFG)`.

In PyBEL, we recognize that these modifications are actually annotations to the type of relationship between the subject's entity and the object's entity. `p(hgnc:ABC) -> tloc(p(hgnc:YFG), GO:intracellular, GO:"cell membrane")` is about the relationship between `p(hgnc:ABC)` and `p(hgnc:YFG)`, while the information about the translocation qualifies that the object is undergoing an event, and not just the abundance. This is a confusion with the use of `proteinAbundance` as a keyword, and perhaps is why many people prefer to use just the keyword `p`

3.5.2 Example Edge Data Structure

Because this data is associated with an edge, the node data for the subject and object are not included explicitly. However, information about the activities, modifiers, and transformations on the subject and object are included. Below is the “skeleton” for the edge data model in PyBEL:

```
from pybel.constants import *

{
    SUBJECT: {
        # ... modifications to the subject node. Only present if non-empty.
    },
    RELATION: POSITIVE_CORRELATION,
    OBJECT: {
        # ... modifications to the object node. Only present if non-empty.
    },
    EVIDENCE: ...,
    CITATION : {
        CITATION_TYPE: CITATION_TYPE_PUBMED,
        CITATION_REFERENCE: ...,
        CITATION_DATE: 'YYYY-MM-DD',
        CITATION_AUTHORS: 'Jon Snow|John Doe',
    },
    ANNOTATIONS: {
        'Disease': {
            'Colorectal Cancer': True,
        },
        # ... additional annotations as tuple[str,dict[str,bool]] pairs
    },
}
```

Each edge must contain the `RELATION`, `EVIDENCE`, and `CITATION` entries. The `CITATION` must minimally contain `CITATION_TYPE` and `CITATION_REFERENCE` since these can be used to look up additional metadata.

Note: Since version 0.10.2, annotations now always appear as dictionaries, even if only one value is present.

3.5.3 Activities

Modifiers are added to this structure as well. Under this schema, `p(hgnc:GSK3B, pmod(P, S, 9)) pos act(p(hgnc:GSK3B), ma(kin))` becomes:

```
from pybel.constants import *

{
    RELATION: POSITIVE_CORRELATION,
    OBJECT: {
        MODIFIER: ACTIVITY,
        EFFECT: {
            NAME: 'kin',
            NAMESPACE: BEL_DEFAULT_NAMESPACE,
        }
    },
    CITATION: { ... },
    EVIDENCE: ...,
    ANNOTATIONS: { ... },
}
```

Activities without molecular activity annotations do not contain an `pybel.constants.EFFECT` entry: Under this schema, `p(hgnc:GSK3B, pmod(P, S, 9)) pos act(p(hgnc:GSK3B))` becomes:

```
from pybel.constants import *

{
    RELATION: POSITIVE_CORRELATION,
    OBJECT: {
        MODIFIER: ACTIVITY
    },
    CITATION: { ... },
    EVIDENCE: ...,
    ANNOTATIONS: { ... },
}
```

3.5.4 Locations

Locations.

Location data also is added into the information in the edge for the node (subject or object) for which it was annotated. `p(HGNC:GSK3B, pmod(P, S, 9), loc(GO:lysosome)) pos act(p(HGNC:GSK3B), ma(kin))` becomes:

```
from pybel.constants import *

{
    SUBJECT: {
        LOCATION: {
            NAMESPACE: 'GO',
            NAME: 'lysosome',
        }
    },
    RELATION: POSITIVE_CORRELATION,
```

(continues on next page)

(continued from previous page)

```

OBJECT: {
    MODIFIER: ACTIVITY,
    EFFECT: {
        NAMESPACE: BEL_DEFAULT_NAMESPACE
        NAME: 'kin',
    }
},
EVIDENCE: ...,
CITATION: { ... },
}

```

The addition of the `location()` element in BEL 2.0 allows for the unambiguous expression of the differences between the process of hypothetical HGNC:A moving from one place to another and the existence of hypothetical HGNC:A in a specific location having different effects. In BEL 1.0, this action had its own node, but this introduced unnecessary complexity to the network and made querying more difficult. This calls for thoughtful consideration of the following two statements:

- `tloc(p(HGNC:A), fromLoc(GO:intracellular), toLoc(GO:"cell membrane")) -> p(HGNC:B)`
- `p(HGNC:A, location(GO:"cell membrane")) -> p(HGNC:B)`

See also:

- BEL 2.0 specification on [cellular location \(2.2.4\)](#)
- PyBEL module `pybel.parser.modifiers.get_location_language`

3.5.5 Translocations

Translocations have their own unique syntax. `p(hgnc:YFG1) -> sec(p(hgnc:YFG2))` becomes:

```

from pybel.constants import *

{
    RELATION: INCREASES,
    OBJECT: {
        MODIFIER: TRANSLOCATION,
        EFFECT: {
            FROM_LOC: {
                NAMESPACE: 'GO',
                NAME: 'intracellular',
            },
            TO_LOC: {
                NAMESPACE: 'GO',
                NAME: 'extracellular space',
            }
        }
    },
    CITATION: { ... },
    EVIDENCE: ...,
    ANNOTATIONS: { ... },
}

```

See also:

BEL 2.0+ tutorial on [translocations](#)

3.5.6 Degradations

Degradations are more simple, because there's no `:pybel.constants.EFFECT` entry. `p(hgnc:YFG1) -> deg(p(hgnc:YFG2))` becomes:

```
from pybel.constants import *

{
    RELATION: INCREASES,
    OBJECT: {
        MODIFIER: DEGRADATION,
    },
    CITATION: { ... },
    EVIDENCE: ...,
    ANNOTATIONS: { ... },
}
```

Warning: Degradations only provide syntax sugar and will be automatically upgraded in a future version of PyBEL such that:

- `deg(X) -> Y` is upgraded to `X -| Y`
- `deg(X) -| Y` is upgraded to `X -> Y`
- `deg(X) => Y` is upgraded to `X =| Y`
- `deg(X) cnc Y` is upgraded to `X cnc Y`
- `X -> deg(Y)` is upgraded to `X -| Y`
- `X => deg(Y)` is upgraded to `X =| Y`
- `X cnc deg(Y)` is upgraded to `X cnc Y`
- `X -| deg(Y)` is undefined

EXAMPLE NETWORKS

This directory contains example networks, precompiled as BEL graphs that are appropriate to use in examples.

An example describing EGF's effect on cellular processes.

```
SET Citation = {"PubMed","Clin Cancer Res 2003 Jul 9(7) 2416-25","12855613"}
SET Evidence = "This induction was not seen either when LNCaP cells were treated with
↳flutamide or conditioned medium were pretreated with antibody to the epidermal growth
↳factor (EGF)"
SET Species = 9606

tscript(p(HGNC:AR)) increases p(HGNC:EGF)

UNSET ALL

SET Citation = {"PubMed","Int J Cancer 1998 Jul 3 77(1) 138-45","9639405"}
SET Evidence = "DU-145 cells treated with 5000 U/ml of IFNgamma and IFN alpha, both
↳reduced EGF production with IFN gamma reduction more significant."
SET Species = 9606

p(HGNC:IFNA1) decreases p(HGNC:EGF)
p(HGNC:IFNG) decreases p(HGNC:EGF)

UNSET ALL

SET Citation = {"PubMed","DNA Cell Biol 2000 May 19(5) 253-63","10855792"}
SET Evidence = "Although found predominantly in the cytoplasm and, less abundantly, in
↳the nucleus, VCP can be translocated from the nucleus after stimulation with epidermal
↳growth factor."
SET Species = 9606

p(HGNC:EGF) increases tloc(p(HGNC:VCP), G0:nucleus, G0:cytoplasm)

UNSET ALL

SET Citation = {"PubMed","J Clin Oncol 2003 Feb 1 21(3) 447-52","12560433"}
SET Evidence = "Valosin-containing protein (VCP; also known as p97) has been shown to be
↳associated with antiapoptotic function and metastasis via activation of the nuclear
↳factor-kappaB signaling pathway."
SET Species = 9606
```

(continues on next page)

(continued from previous page)

```
cat(p(HGNC:VCP)) increases tscript(complex(p(HGNC:NFKB1), p(HGNC:NFKB2), p(HGNC:REL),
↳p(HGNC:RELA), p(HGNC:RELB)))
tscript(complex(p(HGNC:NFKB1), p(HGNC:NFKB2), p(HGNC:REL), p(HGNC:RELA), p(HGNC:RELB)))
↳decreases bp(MESHPP:Apoptosis)
```

```
UNSET ALL
```

pybel.examples.egf_graph

Curation of the article “Genetics ignite focus on microglial inflammation in Alzheimer’s disease”.

```
SET Citation = {"PubMed", "26438529"}
SET Evidence = "Sialic acid binding activates CD33, resulting in phosphorylation of the
↳CD33
immunoreceptor tyrosine-based inhibitory motif (ITIM) domains and activation of the SHP-
↳1 and
SHP-2 tyrosine phosphatases [66, 67]."
```

```
complex(p(HGNC:CD33),a(CHEBI:"sialic acid")) -> p(HGNC:CD33, pmod(P))
act(p(HGNC:CD33, pmod(P))) => act(p(HGNC:PTPN6), ma(phos))
act(p(HGNC:CD33, pmod(P))) => act(p(HGNC:PTPN11), ma(phos))
```

```
UNSET {Evidence, Species}
```

```
SET Evidence = "These phosphatases act on multiple substrates, including Syk, to inhibit
↳immune
activation [68, 69]. Hence, CD33 activation leads to increased SHP-1 and SHP-2 activity
↳that antagonizes Syk,
inhibiting ITAM-signaling proteins, possibly including TREM2/DAP12 (Fig. 1, [70, 71])."
```

```
SET Species = 9606
```

```
act(p(HGNC:PTPN6)) =| act(p(HGNC:SYK))
act(p(HGNC:PTPN11)) =| act(p(HGNC:SYK))
act(p(HGNC:SYK)) -> act(p(HGNC:TREM2))
act(p(HGNC:SYK)) -> act(p(HGNC:TYROBP))
```

```
UNSET ALL
```

pybel.examples.sialic_acid_graph

An example describing a single evidence about BRAF.

```
SET Citation = {"PubMed", "11283246"}
SET Evidence = "Expression of both dominant negative forms, RasN17 and Rap1N17, in UT7-
↳Mpl cells decreased
thrombopoietin-mediated Elk1-dependent transcription. This suggests that both Ras and
↳Rap1 contribute to
thrombopoietin-induced ELK1 transcription."
```

```
SET Species = 9606
```

```
p(HGNC:THPO) increases kin(p(HGNC:BRAF))
```

(continues on next page)

(continued from previous page)

```
p(HGNC:THPO) increases kin(p(HGNC:RAF1))
kin(p(HGNC:BRAF)) increases tscript(p(HGNC:ELK1))
```

```
UNSET ALL
```

`pybel.examples.braf_example_graph`

An example describing statins.

`pybel.examples.statin_graph`

An example describing a translocation.

```
SET Citation = {"PubMed", "16170185"}
SET Evidence = "These modifications render Ras functional and capable of localizing to
↳ the lipid-rich inner surface of the cell membrane. The first and most critical
↳ modification, farnesylation, which is principally catalyzed by protein FTase, adds a
↳ 15-carbon hydrobobic farnesyl isoprenyl tail to the carboxyl terminus of Ras."
SET TextLocation = Review

cat(complex(p(HGNC:FNTA),p(HGNC:FNTB))) directlyIncreases p(SFAM:"RAS Family",pmod(F))
p(SFAM:"RAS Family",pmod(F)) directlyIncreases tloc(p(SFAM:"RAS Family"),MESHCS:
↳ "Intracellular Space",MESHCS:"Cell Membrane")
```

`pybel.examples.ras_tloc_graph`

FILTERS

This module contains functions for filtering node and edge iterables.

It relies heavily on the concepts of [functional programming](#) and the concept of [predicates](#).

`pybel.struct.filters.invert_edge_predicate(edge_predicate)`

Build an edge predicate that is the inverse of the given edge predicate.

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.and_edge_predicates(edge_predicates)`

Concatenate multiple edge predicates to a new predicate that requires all predicates to be met.

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.filter_edges(graph, edge_predicates)`

Apply a set of filters to the edges iterator of a BEL graph.

Return type `Iterable[Tuple[BaseEntity, BaseEntity, str]]`

Returns An iterable of edges that pass all predicates

`pybel.struct.filters.count_passed_edge_filter(graph, edge_predicates)`

Return the number of edges passing a given set of predicates.

Return type `int`

`pybel.struct.filters.build_pmid_exclusion_filter(pmids)`

Fail for edges with citations whose references are one of the given PubMed identifiers.

Parameters `pmids` (`Union[str, Iterable[str]]`) – A PubMed identifier or list of PubMed identifiers to filter against

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_annotation_dict_all_filter(annotations)`

Build an edge predicate for edges whose annotations are super-dictionaries of the given dictionary.

If no annotations are given, will always evaluate to true.

Parameters `annotations` (`Mapping[str, Iterable[str]]`) – The annotation query dict to match

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_annotation_dict_any_filter(annotations)`

Build an edge predicate that passes for edges whose data dictionaries match the given dictionary.

If the given dictionary is empty, will always evaluate to true.

Parameters `annotations` (`Mapping[str, Iterable[str]]`) – The annotation query dict to match

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_upstream_edge_predicate(nodes)`

Build an edge predicate that pass for relations for which one of the given nodes is the object.

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_downstream_edge_predicate(nodes)`

Build an edge predicate that passes for edges for which one of the given nodes is the subject.

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_relation_predicate(relations)`

Build an edge predicate that passes for edges with the given relation.

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_pmids_inclusion_filter(pmids)`

Build an edge predicate that passes for edges with citations from the given PubMed identifier(s).

Parameters `pmids` (`Union[str, Iterable[str]]`) – A PubMed identifier or list of PubMed identifiers to filter for

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.build_author_inclusion_filter(authors)`

Build an edge predicate that passes for edges with citations written by the given author(s).

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.edge_predicate(func)`

Decorate an edge predicate function that only takes a dictionary as its singular argument.

Apply this as a decorator to a function that takes a single argument, a PyBEL node data dictionary, to make sure that it can also accept a pair of arguments, a BELGraph and a PyBEL node tuple as well.

Return type `Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]`

`pybel.struct.filters.true_edge_predicate(graph, u, v, k)`

Return true for all edges.

Return type `bool`

`pybel.struct.filters.false_edge_predicate(graph, u, v, k)`

Return false for all edges.

Return type `bool`

`pybel.struct.filters.has_provenance(edge_data)`

Check if the edge has provenance information (i.e. citation and evidence).

Return type `bool`

`pybel.struct.filters.has_pubmed(edge_data)`

Check if the edge has a PubMed citation.

Return type `bool`

`pybel.struct.filters.has_pmc(edge_data)`

Check if the edge has a PMC citation.

Return type `bool`

`pybel.struct.filters.has_authors(edge_data)`

Check if the edge contains author information for its citation.

Return type `bool`

`pybel.struct.filters.is_causal_relation(edge_data)`

Check if the given relation is causal.

Return type `bool`

`pybel.struct.filters.not_causal_relation(edge_data)`

Check if the given relation is not causal.

Return type `bool`

`pybel.struct.filters.is_direct_causal_relation(edge_data)`

Check if the edge is a direct causal relation.

Return type `bool`

`pybel.struct.filters.is_associative_relation(edge_data)`

Check if the edge has an association relation.

Return type `bool`

`pybel.struct.filters.has_polarity(edge_data)`

Check if the edge has polarity.

Return type `bool`

`pybel.struct.filters.edge_has_activity(edge_data)`

Check if the edge contains an activity in either the subject or object.

Return type `bool`

`pybel.struct.filters.edge_has_degradation(edge_data)`

Check if the edge contains a degradation in either the subject or object.

Return type `bool`

`pybel.struct.filters.edge_has_translocation(edge_data)`

Check if the edge has a translocation in either the subject or object.

Return type `bool`

`pybel.struct.filters.edge_has_annotation(edge_data, key)`

Check if an edge has the given annotation.

Parameters

- **edge_data** (`Mapping`) – The data dictionary from a BELGraph’s edge
- **key** (`str`) – An annotation key

Return type `Optional[Any]`

Returns If the annotation key is present in the current data dictionary

For example, it might be useful to print all edges that are annotated with ‘Subgraph’:

```
>>> from pybel.examples import sialic_acid_graph
>>> from pybel.examples.sialic_acid_example import sialic_acid_cd33_complex, cd33
>>> edges = {
...     (u, v)
...     for u, v, data in sialic_acid_graph.edges(data=True)
...     if edge_has_annotation(data, 'Species')
... }
>>> assert (sialic_acid_cd33_complex, cd33) in edges
```

`pybel.struct.filters.has_pathology_causal(graph, u, v, k)`

Check if the subject is a pathology and has a causal relationship with a non bioprocess/pathology.

Return type `bool`

Returns If the subject of this edge is a pathology and it participates in a causal reaction.

`pybel.struct.filters.filter_nodes(graph, node_predicates)`

Apply a set of predicates to the nodes iterator of a BEL graph.

Return type `Iterable[BaseEntity]`

`pybel.struct.filters.get_nodes(graph, node_predicates)`

Get the set of all nodes that pass the predicates.

Return type `Set[BaseEntity]`

`pybel.struct.filters.count_passed_node_filter(graph, node_predicates)`

Count how many nodes pass a given set of node predicates.

Return type `int`

`pybel.struct.filters.summarize_node_filter(graph, node_filters)`

Print a summary of the number of nodes passing a given set of filters.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **node_filters** (`Union[Callable[[BELGraph, BaseEntity], bool], Iterable[Callable[[BELGraph, BaseEntity], bool]]]) – A node filter or list/tuple of node filters`

Return type `None`

`pybel.struct.filters.get_nodes_by_function(graph, func)`

Get all nodes with the given function(s).

Return type `Set[BaseEntity]`

`pybel.struct.filters.get_nodes_by_namespace(graph, namespaces)`

Get all nodes identified by the given namespace(s).

Return type `Set[BaseEntity]`

`pybel.struct.filters.function_inclusion_filter_builder(func)`

Build a filter that only passes on nodes of the given function(s).

Parameters **func** (`Union[str, Iterable[str]]`) – A BEL Function or list/set/tuple of BEL functions

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.function_exclusion_filter_builder(func)`

Build a filter that fails on nodes of the given function(s).

Parameters **func** (`Union[str, Iterable[str]]`) – A BEL Function or list/set/tuple of BEL functions

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.data_missing_key_builder(key)`

Build a filter that passes only on nodes that don't have the given key in their data dictionary.

Parameters **key** (`str`) – A key for the node's data dictionary

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.build_node_data_search(key, data_predicate)`

Build a filter for nodes whose associated data with the given key passes the given predicate.

Parameters

- **key** (`Union[str, List[str]]`) – The node data dictionary key to check
- **data_predicate** (`Callable[[Any], bool]`) – The filter to apply to the node data dictionary

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.build_node_graph_data_search(key, data_predicate)`

Build a function for testing data associated with the node in the graph.

Parameters

- **key** (`Union[str, List[str]]`) – The node data dictionary key to check
- **data_predicate** (`Callable[[Any], bool]`) – The filter to apply to the node data dictionary

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.build_node_key_search(query, key)`

Build a node filter for nodes whose values for the given key are superstrings of the query string(s).

Parameters

- **query** (`Union[str, Iterable[str]]`) – The query string or strings to check if they're in the node name
- **key** (`Union[str, List[str]]`) – The key for the node data dictionary. Should refer only to entries that have str values

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.build_node_name_search(query)`

Search nodes' names.

Is a thin wrapper around `build_node_key_search()` with `pybel.constants.NAME`

Parameters **query** (`Union[str, Iterable[str]]`) – The query string or strings to check if they're in the node name

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.namespace_inclusion_builder(namespace)`

Build a predicate for namespace inclusion.

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.has_activity(graph, node)`

Return true if over any of the node's edges, it has a molecular activity.

Return type `bool`

`pybel.struct.filters.has_edge_modifier(graph, node, modifier)`

Return true if over any of a nodes edges, it has a given modifier.

Modifier can be one of:

- `pybel.constants.ACTIVITY`,
- `pybel.constants.DEGRADATION`
- `pybel.constants.TRANSLLOCATION`.

Parameters **modifier** (*str*) – One of `pybel.constants.ACTIVITY`, `pybel.constants.Degradation`, or `pybel.constants.Translocation`

Return type `bool`

`pybel.struct.filters.is_degraded(graph, node)`
Return true if over any of the node's edges, it is degraded.

Return type `bool`

`pybel.struct.filters.is_translocated(graph, node)`
Return true if over any of the node's edges, it is translocated.

Return type `bool`

`pybel.struct.filters.is_isolated_list_abundance(graph, node, cls=<class
'pybel.dsl.node_classes.ListAbundance'>)`
Return if the node is a list abundance but has no qualified edges.

Return type `bool`

`pybel.struct.filters.none_of(nodes)`
Build a node predicate that returns false for the given nodes.

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.one_of(nodes)`
Build a function that returns true for the given nodes.

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.has_fragment(node, variant_cls)`
Return true if the node has at least one of the given variant.

Return type `bool`

`pybel.struct.filters.has_gene_modification(node, variant_cls)`
Return true if the node has at least one of the given variant.

Return type `bool`

`pybel.struct.filters.has_hgvs(node, variant_cls)`
Return true if the node has at least one of the given variant.

Return type `bool`

`pybel.struct.filters.has_protein_modification(node, variant_cls)`
Return true if the node has at least one of the given variant.

Return type `bool`

`pybel.struct.filters.has_variant(node)`
Return true if the node has any variants.

Return type `bool`

`pybel.struct.filters.has_causal_edges(graph, node)`
Check if the node has any causal out-edges or in-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.has_causal_in_edges(graph, node)`

Check if the node has any causal in-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.has_causal_out_edges(graph, node)`

Check if the node has any causal out-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.has_in_edges(graph, node, edge_types)`

Check if the node has any in-edges in the given set.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term
- **edge_types** (`Set[str]`) – A collection of edge types to check against

Return type `bool`

`pybel.struct.filters.has_out_edges(graph, node, edge_types)`

Check if the node has any out-edges in the given set.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term
- **edge_types** (`Set[str]`) – A collection of edge types to check against

Return type `bool`

`pybel.struct.filters.is_causal_central(graph, node)`

Check if the node has both causal in-edges and also causal out-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.is_causal_sink(graph, node)`

Check if the node has causal in-edges but no causal out-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.is_causal_source(graph, node)`

Check if the node has causal out-edges but no causal in-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.no_causal_edges(graph, node)`

Check if the node does not have any causal out-edges or in-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.no_causal_in_edges(graph, node)`

Check if the node does not have any causal in-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.no_causal_out_edges(graph, node)`

Check if the node does not have any causal out-edges.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term

Return type `bool`

`pybel.struct.filters.no_in_edges(graph, node, edge_types)`

Check if the node does not have any in-edges in the given set.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term
- **edge_types** (*Set[str]*) – A collection of edge types to check against

Return type `bool`

`pybel.struct.filters.no_out_edges(graph, node, edge_types)`

Check if the node does not have any out-edges in the given set.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node** (*BaseEntity*) – A BEL term
- **edge_types** (*Set[str]*) – A collection of edge types to check against

Return type `bool`

`pybel.struct.filters.concatenate_node_predicates(node_predicates)`

Concatenate multiple node predicates to a new predicate that requires all predicates to be met.

Example usage:

```
>>> from pybel import BELGraph
>>> from pybel.dsl import Protein
>>> from pybel.struct.filters import not_gene, not_rna
>>> app_protein = Protein(name='APP', namespace='hgnc', identifier='620')
>>> app_rna = app_protein.get_rna()
>>> app_gene = app_rna.get_gene()
>>> graph = BELGraph()
>>> _ = graph.add_transcription(app_gene, app_rna)
>>> _ = graph.add_translation(app_rna, app_protein)
>>> node_predicate = concatenate_node_predicates([not_rna, not_gene])
>>> assert node_predicate(graph, app_protein)
>>> assert not node_predicate(graph, app_rna)
>>> assert not node_predicate(graph, app_gene)
```

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.false_node_predicate(_)`

Return false for all nodes.

Return type `bool`

`pybel.struct.filters.invert_node_predicate(f)`

Build a node predicate that is the inverse of the given node predicate.

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.node_predicate(f)`

Tag a node predicate that takes a dictionary to also accept a pair of (*BELGraph*, node).

Apply this as a decorator to a function that takes a single argument, a PyBEL node, to make sure that it can also accept a pair of arguments, a *BELGraph* and a PyBEL node as well.

Return type `Callable[[BELGraph, BaseEntity], bool]`

`pybel.struct.filters.true_node_predicate(_)`

Return true for all nodes.

Given BEL graph *graph*, applying `true_predicate()` with a predicate on the nodes iterable as in `filter(keep_node_permissive, graph)` will result in the same iterable as iterating directly over a *BELGraph*

Return type `bool`

`pybel.struct.filters.part_has_modifier(edge_data, part, modifier)`

Return true if the modifier is in the given subject/object part.

Parameters

- **edge_data** (`Mapping`) – PyBEL edge data dictionary
- **part** (`str`) – either `pybel.constants.SUBJECT` or `pybel.constants.OBJECT`
- **modifier** (`str`) – The modifier to look for

Return type `bool`

GROUPING

Functions for grouping BEL graphs into sub-graphs.

`pybel.struct.grouping.get_subgraphs_by_annotation(graph, annotation, sentinel=None)`

Stratify the given graph into sub-graphs based on the values for edges' annotations.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group by
- **sentinel** (*Optional[str]*) – The value to stick unannotated edges into. If none, does not keep undefined.

Return type *Mapping[Entity, BELGraph]*

`pybel.struct.grouping.get_subgraphs_by_citation(graph)`

Stratify the graph based on citations.

Parameters **graph** (*BELGraph*) – A BEL graph

Return type *Mapping[Tuple[str, str], BELGraph]*

Returns A mapping of each citation db/id to the BEL graph from it.

OPERATIONS

This page outlines operations that can be done to BEL graphs.

`pybel.struct.left_full_join(g, h)`

Add all nodes and edges from `h` to `g`, in-place for `g`.

Parameters

- `g` (`pybel.BELGraph`) – A BEL graph
- `h` (`pybel.BELGraph`) – A BEL graph

Example usage:

```
>>> import pybel
>>> g = pybel.from_bel_script('...')
>>> h = pybel.from_bel_script('...')
>>> left_full_join(g, h)
```

Return type `None`

`pybel.struct.left_outer_join(g, h)`

Only add components from the `h` that are touching `g`.

Algorithm:

1. Identify all weakly connected components in `h`
2. Add those that have an intersection with the `g`

Parameters

- `g` (`BELGraph`) – A BEL graph
- `h` (`BELGraph`) – A BEL graph

Example usage:

```
>>> import pybel
>>> g = pybel.from_bel_script('...')
>>> h = pybel.from_bel_script('...')
>>> left_outer_join(g, h)
```

Return type `None`

`pybel.struct.union(graphs, use_tqdm=False)`

Take the union over a collection of graphs into a new graph.

Assumes iterator is longer than 2, but not infinite.

Parameters

- **graphs** (*iter*[[BELGraph](#)]) – An iterator over BEL graphs. Can't be infinite.
- **use_tqdm** (*bool*) – Should a progress bar be displayed?

Returns A merged graph

Return type [BELGraph](#)

Example usage:

```
>>> import pybel
>>> g = pybel.from_bel_script('...')
>>> h = pybel.from_bel_script('...')
>>> k = pybel.from_bel_script('...')
>>> merged = union([g, h, k])
```

PIPELINE

class `pybel.Pipeline(protocol=None)`

Build and runs analytical pipelines on BEL graphs.

Example usage:

```
>>> from pybel import BELGraph
>>> from pybel.struct.pipeline import Pipeline
>>> from pybel.struct.mutation import enrich_protein_and_rna_origins, prune_protein_
    ↪ rna_origins
>>> graph = BELGraph()
>>> example = Pipeline()
>>> example.append(enrich_protein_and_rna_origins)
>>> example.append(prune_protein_rna_origins)
>>> result = example.run(graph)
```

Initialize the pipeline with an optional pre-defined protocol.

Parameters `protocol` (`Optional[Iterable[Dict]]`) – An iterable of dictionaries describing how to transform a network

static from_functions(`functions`)

Build a pipeline from a list of functions.

Parameters `functions` (`iter[(((pybel.BELGraph) -> pybel.BELGraph) or ((pybel.BELGraph) -> None) or str)]`) – A list of functions or names of functions

Example with function:

```
>>> from pybel.struct.pipeline import Pipeline
>>> from pybel.struct.mutation import remove_associations
>>> pipeline = Pipeline.from_functions([remove_associations])
```

Equivalent example with function names:

```
>>> from pybel.struct.pipeline import Pipeline
>>> pipeline = Pipeline.from_functions(['remove_associations'])
```

Lookup by name is possible for built in functions, and those that have been registered correctly using one of the four decorators:

1. `pybel.struct.pipeline.transformation()`,
2. `pybel.struct.pipeline.in_place_transformation()`,
3. `pybel.struct.pipeline.uni_transformation()`,

4. `pybel.struct.pipeline.uni_in_place_transformation()`,

Return type *Pipeline*

append(*name*, **args*, ***kwargs*)

Add a function (either as a reference, or by name) and arguments to the pipeline.

Parameters

- **name** (*str* or (*pybel.BELGraph* -> *pybel.BELGraph*)) – The name of the function
- **args** – The positional arguments to call in the function
- **kwargs** – The keyword arguments to call in the function

Return type *Pipeline*

Returns This pipeline for fluid query building

Raises *MissingPipelineFunctionError* – If the function is not registered

extend(*protocol*)

Add another pipeline to the end of the current pipeline.

Parameters **protocol** (*Union*[*Iterable*[*Dict*], *Pipeline*]) – An iterable of dictionaries (or another Pipeline)

Return type *Pipeline*

Returns This pipeline for fluid query building

Example:

```
>>> p1 = Pipeline.from_functions(['enrich_protein_and_rna_origins']) >>> p2 = Pipeline.from_functions(['remove_pathologies']) >>> p1.extend(p2)
```

run(*graph*, *universe=None*)

Run the contained protocol on a seed graph.

Parameters

- **graph** (*pybel.BELGraph*) – The seed BEL graph
- **universe** (*pybel.BELGraph*) – Allows just-in-time setting of the universe in case it wasn't set before. Defaults to the given network.

Returns The new graph is returned if not applied in-place

Return type *pybel.BELGraph*

to_json()

Return this pipeline as a JSON list.

Return type *List*

dumps(***kwargs*)

Dump this pipeline as a JSON string.

Return type *str*

dump(*file*, ***kwargs*)

Dump this protocol to a file in JSON.

Return type *None*

static from_json(*data*)

Build a pipeline from a JSON list.

Return type *Pipeline*

static `load(file)`

Load a protocol from JSON contained in file.

Return type `Pipeline`

Returns The pipeline represented by the JSON in the file

Raises `MissingPipelineFunctionError` – If any functions are not registered

static `loads(s)`

Load a protocol from a JSON string.

Parameters `s (str)` – A JSON string

Return type `Pipeline`

Returns The pipeline represented by the JSON in the file

Raises `MissingPipelineFunctionError` – If any functions are not registered

static `union(pipelines)`

Take the union of multiple pipelines.

Parameters `pipelines (Iterable[Pipeline])` – A list of pipelines

Return type `Pipeline`

Returns The union of the results from multiple pipelines

static `intersection(pipelines)`

Take the intersection of the results from multiple pipelines.

Parameters `pipelines (Iterable[Pipeline])` – A list of pipelines

Return type `Pipeline`

Returns The intersection of results from multiple pipelines

8.1 Transformation Decorators

This module contains the functions for decorating transformation functions.

A transformation function takes in a `pybel.BELGraph` and either returns `None` (in-place) or a new `pybel.BELGraph` (out-of-place).

`pybel.struct.pipeline.decorators.in_place_transformation(func)`

A decorator for functions that modify BEL graphs in-place

`pybel.struct.pipeline.decorators.uni_in_place_transformation(func)`

A decorator for functions that require a “universe” graph and modify BEL graphs in-place

`pybel.struct.pipeline.decorators.uni_transformation(func)`

A decorator for functions that require a “universe” graph and create new BEL graphs from old BEL graphs

`pybel.struct.pipeline.decorators.transformation(func)`

A decorator for functions that create new BEL graphs from old BEL graphs

`pybel.struct.pipeline.decorators.get_transformation(name)`

Get a transformation function and error if its name is not registered.

Parameters `name (str)` – The name of a function to look up

Returns A transformation function

Raises `MissingPipelineFunctionError` – If the given function name is not registered

8.2 Exceptions

Exceptions for the `pybel.struct.pipeline` module.

exception `pybel.struct.pipeline.exc.MissingPipelineFunctionError`

Raised when trying to run the pipeline with a function that isn't registered.

exception `pybel.struct.pipeline.exc.MetaValueError`

Raised when getting an invalid meta value.

exception `pybel.struct.pipeline.exc.MissingUniverseError`

Raised when running a universe function without a universe being present.

exception `pybel.struct.pipeline.exc.DeprecationMappingError`

Raised when applying the deprecation function annotation and the given name already is being used.

exception `pybel.struct.pipeline.exc.PipelineNameError`

Raised when a second function tries to use the same name.

QUERY

Query builder for PyBEL.

exception `pybel.struct.query.QueryMissingNetworksError`

Raised if a query is created from json but doesn't have a listing of network identifiers.

exception `pybel.struct.query.NodeDegreeIterError`

Raised when failing to iterate over node degrees.

class `pybel.struct.query.Query(network_ids=None, seeding=None, pipeline=None)`

Represents a query over a network store.

Build a query.

Parameters `network_ids` (`Union[None, int, Iterable[int]]`) – Database network identifiers identifiers

append_network(`network_id`)

Add a network to this query.

Parameters `network_id` (`int`) – The database identifier of the network

Return type `Query`

Returns self for fluid API

append_seeding_induction(`nodes`)

Add a seed induction method.

Return type `Seeding`

Returns seeding container for fluid API

append_seeding_neighbors(`nodes`)

Add a seed by neighbors.

Return type `Seeding`

Returns seeding container for fluid API

append_seeding_annotation(`annotation, values`)

Add a seed induction method for single annotation's values.

Parameters

- **annotation** (`str`) – The annotation to filter by
- **values** (`Set[str]`) – The values of the annotation to keep

Return type `Seeding`

append_seeding_sample(**kwargs)

Add seed induction methods.

Kwargs can have `number_edges` or `number_seed_nodes`.

Return type *Seeding*

append_pipeline(name, *args, **kwargs)

Add an entry to the pipeline. Defers to `pybel_tools.pipeline.Pipeline.append()`.

Parameters **name** (*str* or *types.FunctionType*) – The name of the function

Return type *Pipeline*

Returns This pipeline for fluid query building

run(manager)

Run this query and returns the resulting BEL graph.

Parameters **manager** – A cache manager

Return type Optional[*pybel.BELGraph*]

to_json()

Return this query as a JSON object.

Return type *Dict*

dump(file, **kwargs)

Dump this query to a file as JSON.

Return type *None*

dumps(**kwargs)

Dump this query to a string as JSON.

Return type *str*

static from_json(data)

Load a query from a JSON dictionary.

Parameters **data** (*Mapping*) – A JSON dictionary

Raises *QueryMissingNetworksError*

Return type *Query*

static load(file)

Load a query from a JSON file.

Raises *QueryMissingNetworksError*

Return type *Query*

static loads(s)

Load a query from a JSON string.

Parameters **s** (*str*) – A stringified JSON query

Raises *QueryMissingNetworksError*

Return type *Query*

class `pybel.struct.query.Seeding`(initlist=None)

Represents a container of seeding methods to apply to a network.

append_induction(nodes)

Add a seed induction method.

Parameters **nodes** (`Union[BaseEntity, List[BaseEntity], List[Dict]]`) – A node or list of nodes

Return type *Seeding*

Returns self for fluid API

append_neighbors(*nodes*)

Add a seed by neighbors.

Parameters **nodes** (`Union[BaseEntity, List[BaseEntity], List[Dict]]`) – A node or list of nodes

Return type *Seeding*

Returns self for fluid API

append_annotation(*annotation, values*)

Add a seed induction method for single annotation's values.

Parameters

- **annotation** (`str`) – The annotation to filter by
- **values** (`Set[str]`) – The values of the annotation to keep

Return type *Seeding*

Returns self for fluid API

append_sample(***kwargs*)

Add seed induction methods.

Kwargs can have `number_edges` or `number_seed_nodes`.

Return type *Seeding*

Returns self for fluid API

run(*graph*)

Seed the graph or return none if not possible.

Return type `Optional[pybel.BELGraph]`

to_json()

Serialize this seeding container to a JSON object.

Return type `List[Dict]`

dump(*file, sort_keys=True, **kwargs*)

Dump this seeding container to a file as JSON.

Return type `None`

dumps(*sort_keys=True, **kwargs*)

Dump this query to a string as JSON.

Return type `str`

static from_json(*data*)

Build a seeding container from a JSON list.

Return type *Seeding*

static load(*file*)

Load a seeding container from a JSON file.

Return type *Seeding*

static loads(*s*)

Load a seeding container from a JSON string.

Return type *Seeding*

`pybel.struct.query.get_subgraph(graph, seed_method=None, seed_data=None, expand_nodes=None, remove_nodes=None)`

Run a pipeline query on graph with multiple sub-graph filters and expanders.

Order of Operations:

1. Seeding by given function name and data
2. Add nodes
3. Remove nodes

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **seed_method** (`Optional[str]`) – The name of the `get_subgraph_by_*` function to use
- **seed_data** (`Optional[Any]`) – The argument to pass to the `get_subgraph` function
- **expand_nodes** (`Optional[List[BaseEntity]]`) – Add the neighborhoods around all of these nodes
- **remove_nodes** (`Optional[List[BaseEntity]]`) – Remove these nodes and all of their in/out edges

Return type `Optional[pybel.BELGraph]`

SUMMARY

Summary functions for BEL graphs.

`pybel.struct.summary.iter_annotation_value_pairs(graph)`

Iterate over the key/value pairs, with duplicates, for each annotation used in a BEL graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Iterable[Tuple[str, Entity]]`

`pybel.struct.summary.iter_annotation_values(graph, annotation)`

Iterate over all of the values for an annotation used in the graph.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `annotation` (`str`) – The annotation to grab

Return type `Iterable[Entity]`

`pybel.struct.summary.get_annotation_values_by_annotation(graph)`

Get the set of values for each annotation used in a BEL graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Mapping[str, Set[Entity]]`

Returns A dictionary of {annotation key: set of annotation values}

`pybel.struct.summary.get_annotation_values(graph, annotation)`

Get all values for the given annotation.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `annotation` (`str`) – The annotation to summarize

Return type `Set[Entity]`

Returns A set of all annotation values

`pybel.struct.summary.count_relations(graph)`

Return a histogram over all relationships in a graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Counter`

Returns A Counter from {relation type: frequency}

`pybel.struct.summary.get_annotations(graph)`

Get the set of annotations used in the graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of annotation keys

`pybel.struct.summary.count_annotations(graph)`

Count how many times each annotation is used in the graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Counter`

Returns A Counter from {annotation key: frequency}

`pybel.struct.summary.get_unused_annotations(graph)`

Get the set of all annotations that are defined in a graph, but are never used.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of annotations

`pybel.struct.summary.get_unused_list_annotation_values(graph)`

Get all of the unused values for list annotations.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Mapping[str, Set[str]]`

Returns A dictionary of {str annotation: set of str values that aren't used}

`pybel.struct.summary.get_metaedge_to_key(graph)`

Get all edge types.

Return type `Mapping[Tuple[str, Optional[Tuple], Optional[Tuple]], Set[Tuple[BaseEntity, BaseEntity, str]]]`

`pybel.struct.summary.iter_sample_metaedges(graph)`

Iterate sampled metaedges.

`pybel.struct.summary.get_syntax_errors(graph)`

List the syntax errors encountered during compilation of a BEL script.

Return type `List[Tuple[Optional[str], BELParserWarning, Mapping]]`

`pybel.struct.summary.count_error_types(graph)`

Count the occurrence of each type of error in a graph.

Return type `Counter[str]`

Returns A Counter of {error type: frequency}

`pybel.struct.summary.count_naked_names(graph)`

Count the frequency of each naked name (names without namespaces).

Return type `Counter[str]`

Returns A Counter from {name: frequency}

`pybel.struct.summary.get_naked_names(graph)`

Get the set of naked names in the graph.

Return type `Set[str]`

`pybel.struct.summary.calculate_incorrect_name_dict(graph)`

Get missing names grouped by namespace.

Return type `Mapping[str, List[str]]`

`pybel.struct.summary.calculate_error_by_annotation(graph, annotation)`

Group error names by a given annotation.

Return type `Mapping[str, List[str]]`

`pybel.struct.summary.get_functions(graph)`

Get the set of all functions used in this graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of functions

`pybel.struct.summary.count_functions(graph)`

Count the frequency of each function present in a graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Counter[str]`

Returns A Counter from {function: frequency}

`pybel.struct.summary.get_namespaces(graph)`

Get the set of all namespaces used in this graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of namespaces

`pybel.struct.summary.count_namespaces(graph)`

Count the frequency of each namespace across all nodes (that have namespaces).

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Counter[str]`

Returns A Counter from {namespace: frequency}

`pybel.struct.summary.get_unused_namespaces(graph)`

Get the set of all namespaces that are defined in a graph, but are never used.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of namespaces that are included but not used

`pybel.struct.summary.count_names_by_namespace(graph, namespace)`

Get the set of all of the names in a given namespace that are in the graph.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `namespace` (`str`) – A namespace prefix

Return type `Counter[str]`

Returns A counter from {name: frequency}

Raises `IndexError` – if the namespace is not defined in the graph.

`pybel.struct.summary.get_names(graph)`

Get all names for each namespace.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Mapping[str, Set[str]]`

`pybel.struct.summary.get_names_by_namespace(graph, namespace)`

Get the set of all of the names in a given namespace that are in the graph.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **namespace** (*str*) – A namespace prefix

Return type `Set[str]`

Returns A set of names belonging to the given namespace that are in the given graph

Raises `IndexError` – if the namespace is not defined in the graph.

`pybel.struct.summary.iterate_node_entities(node)`

Iterate over all named entities that comprise a node.

This includes the node's name, the members/reactants/products of the node, the fusion partners, the named variants, and all recursive ones too.

Parameters `node` (*BaseEntity*) – A BEL node

Entities in a simple protein:

```
>>> from pybel.dsl import Protein
>>> from pybel.language import Entity
>>> from pybel.struct.summary import iterate_entities
>>> protein = Protein(namespace='hgnc', identifier='1455', name='CALR')
>>> protein_entities = list(iterate_node_entities(protein))
>>> assert [Entity(namespace='hgnc', identifier='1455', name='CALR')] == protein_
↳ entities
```

Entities in a protein complex:

```
>>> from pybel.dsl import Protein, ComplexAbundance
>>> from pybel.language import Entity
>>> from pybel.struct.summary import iterate_entities
>>> protein_1 = Protein(namespace='hgnc', identifier='1')
>>> protein_2 = Protein(namespace='hgnc', identifier='2')
>>> complex_1 = ComplexAbundance([protein_1, protein_2])
>>> complex_entities = list(iterate_node_entities(complex_1))
>>> assert [Entity(namespace='hgnc', identifier='1'), Entity(namespace='hgnc',
↳ identifier='2')] == complex_entities
```

Return type `Iterable[Entity]`

`pybel.struct.summary.iterate_entities(graph)`

Iterate over all entities in the graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Iterable[Entity]`

`pybel.struct.summary.node_is_grounded(node)`

Check if a node is grounded.

Parameters `node` (*BELNode*) – A BEL node

Return type `bool`

`pybel.struct.summary.get_ungrounded_nodes(graph)`

Get all ungrounded nodes in the graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[BELNode]`

`pybel.struct.summary.count_variants(graph)`

Count how many of each type of variant a graph has.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Counter[str]`

`pybel.struct.summary.count_pathologies(graph)`

Count the number of edges in which each pathology is incident.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Counter[BELNode]`

`pybel.struct.summary.get_top_pathologies(graph, n=15)`

Get the top highest relationship-having edges in the graph by BEL.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `n` (*Optional[int]*) – The number of top connected pathologies to return. If None, returns all nodes

Return type `List[Tuple[BELNode, int]]`

`pybel.struct.summary.get_top_hubs(graph, *, n=15)`

Get the top hubs in the graph by BEL.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `n` (*Optional[int]*) – The number of top hubs to return. If None, returns all nodes

Return type `List[Tuple[BELNode, int]]`

`pybel.struct.summary.iterate_pubmed_identifiers(graph)`

Iterate over all PubMed identifiers in a graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Iterable[str]`

Returns An iterator over the PubMed identifiers in the graph

`pybel.struct.summary.iterate_pmc_identifiers(graph)`

Iterate over all PMC identifiers in a graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Iterable[str]`

Returns An iterator over the PMC identifiers in the graph

`pybel.struct.summary.get_pubmed_identifiers(graph)`

Get the set of all PubMed identifiers cited in the construction of a graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of all PubMed identifiers cited in the construction of this graph

`pybel.struct.summary.get_pmc_identifiers(graph)`

Get the set of all PMC identifiers cited in the construction of a graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Set[str]`

Returns A set of all PMC identifiers cited in the construction of this graph

Misc. getters.

`pybel.struct.getters.get_tf_pairs(graph, direct_only=False)`

Iterate pairs of `p(X)` and `r(Y)` such that `complex(p(X), g(Y)) -> r(Y)`.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **direct_only** (`bool`) – If true, only uses `directlyIncreases` and `directlyDecreases` relations. Otherwise, allows indirect relations.

Return type `Iterable[Tuple[Protein, Rna, int]]`

MUTATIONS

General-use induction functions.

`pybel.struct.mutation.utils.remove_isolated_nodes(graph)`

Remove isolated nodes from the network, in place.

Parameters `graph` (`pybel.BELGraph`) – A BEL graph

`pybel.struct.mutation.utils.remove_isolated_nodes_op(graph)`

Build a new graph excluding the isolated nodes.

Parameters `graph` (`pybel.BELGraph`) – A BEL graph

Return type `pybel.BELGraph`

`pybel.struct.mutation.utils.expand_by_edge_filter(source, target, edge_predicates)`

Expand a target graph by edges in the source matching the given predicates.

Parameters

- **source** (`pybel.BELGraph`) – A BEL graph
- **target** (`pybel.BELGraph`) – A BEL graph
- **edge_predicates** (`Union[Callable[[BELGraph, BaseEntity, BaseEntity, str], bool], Iterable[Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]]]`) – An edge predicate or list of edge predicates

Returns A BEL sub-graph induced over the edges passing the given filters

Return type `pybel.BELGraph`

COLLAPSE

Functions for collapsing nodes.

`pybel.struct.mutation.collapse.collapse_pair(graph, survivor, victim)`

Rewire all edges from the synonymous node to the survivor node, then deletes the synonymous node.

Does not keep edges between the two nodes.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **survivor** (`BaseEntity`) – The BEL node to collapse all edges on the synonym to
- **victim** (`BaseEntity`) – The BEL node to collapse into the surviving node

Return type `None`

`pybel.struct.mutation.collapse.collapse_nodes(graph, survivor_mapping)`

Collapse all nodes in values to the key nodes, in place.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **survivor_mapping** (`Mapping[BaseEntity, Set[BaseEntity]]`) – A dictionary with survivors as their keys, and iterables of the corresponding victims as values.

Return type `None`

`pybel.struct.mutation.collapse.collapse_all_variants(graph)`

Collapse all genes', RNAs', miRNAs', and proteins' variants to their parents.

Parameters **graph** (`pybel.BELGraph`) – A BEL Graph

Return type `None`

`pybel.struct.mutation.collapse.survivors_are_inconsistent(survivor_mapping)`

Check that there's no transitive shit going on.

Return type `Set[BaseEntity]`

`pybel.struct.mutation.collapse.collapse_to_genes(graph)`

Collapse all protein, RNA, and miRNA nodes to their corresponding gene nodes.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

DELETION

Modules supporting deletion and degradation of graphs.

`pybel.struct.mutation.deletion.remove_filtered_edges(graph, edge_predicates=None)`

Remove edges passing the given edge predicates.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **edge_predicates** (`None` or `((pybel.BELGraph, tuple, tuple, int) -> bool)` or `iter[(pybel.BELGraph, tuple, tuple, int) -> bool]`) – A predicate or list of predicates

Returns

`pybel.struct.mutation.deletion.remove_filtered_nodes(graph, node_predicates=None)`

Remove nodes passing the given node predicates.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

`pybel.struct.mutation.deletion.remove_associations(graph)`

Remove all associative relationships from the graph.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

`pybel.struct.mutation.deletion.remove_pathologies(graph)`

Remove pathology nodes from the graph.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

`pybel.struct.mutation.deletion.remove_biological_processes(graph)`

Remove biological process nodes from the graph.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

`pybel.struct.mutation.deletion.remove_isolated_list_abundances(graph)`

Remove isolated list abundances from the graph.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

`pybel.struct.mutation.deletion.remove_non_causal_edges(graph)`

Remove non-causal edges from the graph.

`pybel.struct.mutation.deletion.prune_protein_rna_origins(graph)`

Delete genes that are only connected to one node, their correspond RNA, by a translation edge.

Parameters **graph** (`pybel.BELGraph`) – A BEL graph

EXPANSION

Mutations that expand the graph.

`pybel.struct.mutation.expansion.expand_node_predecessors(universe, graph, node)`

Expand around the predecessors of the given node in the result graph.

Parameters

- **universe** (*BELGraph*) – The graph containing the stuff to add
- **graph** (*BELGraph*) – The graph to add stuff to
- **node** (*BaseEntity*) – A BEL node

`pybel.struct.mutation.expansion.expand_node_successors(universe, graph, node)`

Expand around the successors of the given node in the result graph.

Parameters

- **universe** (*BELGraph*) – The graph containing the stuff to add
- **graph** (*BELGraph*) – The graph to add stuff to
- **node** (*BaseEntity*) – A BEL node

Return type `None`

`pybel.struct.mutation.expansion.expand_node_neighborhood(universe, graph, node)`

Expand around the neighborhoods of the given node in the result graph.

Note: expands complexes' members

Parameters

- **universe** (*BELGraph*) – The graph containing the stuff to add
- **graph** (*BELGraph*) – The graph to add stuff to
- **node** (*BaseEntity*) – A BEL node

Return type `None`

`pybel.struct.mutation.expansion.expand_nodes_neighborhoods(universe, graph, nodes)`

Expand around the neighborhoods of the given node in the result graph.

Parameters

- **universe** (*BELGraph*) – The graph containing the stuff to add
- **graph** (*BELGraph*) – The graph to add stuff to
- **nodes** (*Iterable[BaseEntity]*) – Nodes from the query graph

Return type `None`

`pybel.struct.mutation.expansion.expand_all_node_neighborhoods(universe, graph, filter_pathologies=False)`

Expand the neighborhoods of all nodes in the given graph.

Parameters

- **universe** (`pybel.BELGraph`) – The graph containing the stuff to add
- **graph** (`pybel.BELGraph`) – The graph to add stuff to
- **filter_pathologies** (`bool`) – Should expansion take place around pathologies?

Return type `None`

`pybel.struct.mutation.expansion.expand_internal(universe, graph)`

Expand on edges between entities in the sub-graph that pass the given filters, in place.

Parameters

- **universe** (`BELGraph`) – The full graph
- **graph** (`BELGraph`) – A sub-graph to find the upstream information

Return type `None`

`pybel.struct.mutation.expansion.expand_upstream_causal(universe, graph)`

Add the upstream causal relations to the given sub-graph.

Parameters

- **universe** (`pybel.BELGraph`) – A BEL graph representing the universe of all knowledge
- **graph** (`pybel.BELGraph`) – The target BEL graph to enrich with upstream causal controllers of contained nodes

`pybel.struct.mutation.expansion.expand_downstream_causal(universe, graph)`

Add the downstream causal relations to the given sub-graph.

Parameters

- **universe** (`pybel.BELGraph`) – A BEL graph representing the universe of all knowledge
- **graph** (`pybel.BELGraph`) – The target BEL graph to enrich with upstream causal controllers of contained nodes

INDUCTION

Mutations that induce a sub-graph.

`pybel.struct.mutation.induction.get_subgraph_by_annotation_value(graph, annotation, values)`
Induce a sub-graph over all edges whose annotations match the given key and value.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group by
- **values** (*Union[str, Iterable[str]]*) – The value(s) for the annotation

Return type *BELGraph*

Returns A subgraph of the original BEL graph

`pybel.struct.mutation.induction.get_subgraph_by_annotations(graph, annotations, or_=None)`
Induce a sub-graph given an annotations filter.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **annotations** (*Union[Mapping[str, str], Mapping[str, Set[str]], Mapping[str, List[Entity]]]*) – Annotation filters (match all with `pybel.utils.subdict_matches()`)
- **or** – if True any annotation should be present, if False all annotations should be present in the edge. Defaults to True.

Return type *BELGraph*

Returns A subgraph of the original BEL graph

`pybel.struct.mutation.induction.get_subgraph_by_pubmed(graph, pubmed_identifiers)`
Induce a sub-graph over the edges retrieved from the given PubMed identifier(s).

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or list[str] pubmed_identifiers** (*str*) – A PubMed identifier or list of PubMed identifiers

Return type *pybel.BELGraph*

`pybel.struct.mutation.induction.get_subgraph_by_authors(graph, authors)`
Induce a sub-graph over the edges retrieved publications by the given author(s).

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- or **list[str] authors** (`str`) – An author or list of authors

Return type `pybel.BELGraph`

`pybel.struct.mutation.induction.get_subgraph_by_neighborhood(graph, nodes)`

Get a BEL graph around the neighborhoods of the given nodes.

Returns none if no nodes are in the graph.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **nodes** (`Iterable[BaseEntity]`) – An iterable of BEL nodes

Return type `Optional[BELGraph]`

Returns A BEL graph induced around the neighborhoods of the given nodes

`pybel.struct.mutation.induction.get_nodes_in_all_shortest_paths(graph, nodes, weight=None, remove_pathologies=False)`

Get a set of nodes in all shortest paths between the given nodes.

Thinly wraps `networkx.all_shortest_paths()`.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **nodes** (`Iterable[BaseEntity]`) – The list of nodes to use to use to find all shortest paths
- **weight** (`Optional[str]`) – Edge data key corresponding to the edge weight. If none, uses unweighted search.
- **remove_pathologies** (`bool`) – Should pathology nodes be removed first?

Return type `Set[BaseEntity]`

Returns A set of nodes appearing in the shortest paths between nodes in the BEL graph

Note: This can be trivially parallelized using `networkx.single_source_shortest_path()`

`pybel.struct.mutation.induction.get_subgraph_by_all_shortest_paths(graph, nodes, weight=None, remove_pathologies=False)`

Induce a subgraph over the nodes in the pairwise shortest paths between all of the nodes in the given list.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **nodes** (`Iterable[BaseEntity]`) – A set of nodes over which to calculate shortest paths
- **weight** (`Optional[str]`) – Edge data key corresponding to the edge weight. If None, performs unweighted search
- **remove_pathologies** (`bool`) – Should the pathology nodes be deleted before getting shortest paths?

Returns A BEL graph induced over the nodes appearing in the shortest paths between the given nodes

Return type `Optional[pybel.BELGraph]`

`pybel.struct.mutation.induction.get_random_path(graph)`

Get a random path from the graph as a list of nodes.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type *List[BaseEntity]*

`pybel.struct.mutation.induction.get_graph_with_random_edges(graph, n_edges)`

Build a new graph from a seeding of edges.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `n_edges` (*int*) – Number of edges to randomly select from the given graph

Return type *BELGraph*

`pybel.struct.mutation.induction.get_random_node(graph, node_blacklist, invert_degrees=None)`

Choose a node from the graph with probabilities based on their degrees.

Parameters

- `node_blacklist` (*Set[BaseEntity]*) – Nodes to filter out
- `invert_degrees` (*Optional[bool]*) – Should the degrees be inverted? Defaults to true.

Return type *Optional[BaseEntity]*

`pybel.struct.mutation.induction.get_random_subgraph(graph, number_edges=None,
number_seed_edges=None, seed=None,
invert_degrees=None)`

Generate a random subgraph based on weighted random walks from random seed edges.

Parameters

- `number_edges` (*Optional[int]*) – Maximum number of edges. Defaults to `pybel_tools.constants.SAMPLE_RANDOM_EDGE_COUNT` (250).
- `number_seed_edges` (*Optional[int]*) – Number of nodes to start with (which likely results in different components in large graphs). Defaults to `SAMPLE_RANDOM_EDGE_SEED_COUNT` (5).
- `seed` (*Optional[int]*) – A seed for the random state
- `invert_degrees` (*Optional[bool]*) – Should the degrees be inverted? Defaults to true.

Return type *BELGraph*

`pybel.struct.mutation.induction.get_upstream_causal_subgraph(graph, nbunch)`

Induce a sub-graph from all of the upstream causal entities of the nodes in the nbunch.

Return type *pybel.BELGraph*

`pybel.struct.mutation.induction.get_downstream_causal_subgraph(graph, nbunch)`

Induce a sub-graph from all of the downstream causal entities of the nodes in the nbunch.

Return type *pybel.BELGraph*

`pybel.struct.mutation.induction.get_subgraph_by_edge_filter(graph, edge_predicates=None)`

Induce a sub-graph on all edges that pass the given filters.

Parameters

- `graph` (*BELGraph*) – A BEL graph

- **edge_predicates** (`Union[Callable[[BELGraph, BaseEntity, BaseEntity, str], bool], Iterable[Callable[[BELGraph, BaseEntity, BaseEntity, str], bool]], None)`) – An edge predicate or list of edge predicates

Return type *BELGraph*

Returns A BEL sub-graph induced over the edges passing the given filters

`pybel.struct.mutation.induction.get_subgraph_by_induction(graph, nodes)`

Induce a sub-graph over the given nodes or return None if none of the nodes are in the given graph.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **nodes** (`Iterable[BaseEntity]`) – A list of BEL nodes in the graph

Return type `Optional[BELGraph]`

`pybel.struct.mutation.induction.get_subgraph_by_node_filter(graph, node_predicates)`

Induce a sub-graph on the nodes that pass the given predicate(s).

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **node_predicates** (`Union[Callable[[BELGraph, BaseEntity], bool], Iterable[Callable[[BELGraph, BaseEntity], bool]]]`) – A node predicate or list of node predicates

Return type *BELGraph*

`pybel.struct.mutation.induction.get_largest_component(graph)`

Get the giant component of a graph.

Parameters **graph** (*BELGraph*) – A BEL graph

Return type *BELGraph*

`pybel.struct.mutation.induction.get_causal_subgraph(graph)`

Build a new sub-graph induced over the causal edges.

Parameters **graph** (*BELGraph*) – A BEL graph

Return type *BELGraph*

INDUCTION AND EXPANSION

Functions for building graphs that use both expansion and induction procedures.

`pybel.struct.mutation.induction_expansion.get_multi_causal_upstream(graph, nbunch)`

Get the union of all the 2-level deep causal upstream subgraphs from the nbunch.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **nbunch** (`Union[BaseEntity, Iterable[BaseEntity]]`) – A BEL node or list of BEL nodes

Returns A subgraph of the original BEL graph

Return type `pybel.BELGraph`

`pybel.struct.mutation.induction_expansion.get_multi_causal_downstream(graph, nbunch)`

Get the union of all of the 2-level deep causal downstream subgraphs from the nbunch.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **nbunch** (`Union[BaseEntity, Iterable[BaseEntity]]`) – A BEL node or list of BEL nodes

Returns A subgraph of the original BEL graph

Return type `pybel.BELGraph`

`pybel.struct.mutation.induction_expansion.get_subgraph_by_second_neighbors(graph, nodes, filter_pathologies=False)`

Get a graph around the neighborhoods of the given nodes and expand to the neighborhood of those nodes.

Returns none if none of the nodes are in the graph.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL graph
- **nodes** (`Iterable[BaseEntity]`) – An iterable of BEL nodes
- **filter_pathologies** (`bool`) – Should expansion take place around pathologies?

Returns A BEL graph induced around the neighborhoods of the given nodes

Return type `Optional[pybel.BELGraph]`

INFERENCE

Mutations for inferring new edges in the graph.

`pybel.struct.mutation.inference.enrich_rnas_with_genes(graph)`

Add the corresponding gene node for each RNA/miRNA node and connect them with a transcription edge.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `None`

`pybel.struct.mutation.inference.enrich_proteins_with_rnas(graph)`

Add the corresponding RNA node for each protein node and connect them with a translation edge.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `None`

`pybel.struct.mutation.inference.enrich_protein_and_rna_origins(graph)`

Add the corresponding RNA for each protein then the corresponding gene for each RNA/miRNA.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `None`

`pybel.struct.mutation.inference.infer_child_relations(graph, node)`

Propagate causal relations to children.

Return type `List[str]`

METADATA

Functions to modify the metadata of graphs, their edges, and their nodes.

`pybel.struct.mutation.metadata.strip_annotations(graph)`

Strip all the annotations from a BEL graph.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `None`

`pybel.struct.mutation.metadata.add_annotation_value(graph, annotation, value, strict=True)`

Add the given annotation/value pair to all qualified edges.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `annotation` (`str`) –
- `value` (`str`) –
- `strict` (`bool`) – Should the function ensure the annotation has already been defined?

Return type `None`

`pybel.struct.mutation.metadata.remove_annotation_value(graph, annotation, value)`

Remove the given annotation/value pair to all qualified edges.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `annotation` (`str`) –
- `value` (`str`) –

Return type `None`

`pybel.struct.mutation.metadata.remove_extra_citation_metadata(graph)`

Remove superfluous metadata associated with a citation (that isn't the db/id).

Best practice is to add this information programmatically.

Return type `None`

INPUT AND OUTPUT

Input and output functions for BEL graphs.

PyBEL provides multiple lossless interchange options for BEL. Lossy output formats are also included for convenient export to other programs. Notably, a *de facto* interchange using Resource Description Framework (RDF) to match the ability of other existing software is excluded due the immaturity of the BEL to RDF mapping.

`pybel.load(path, **kwargs)`
Read a BEL graph.

Parameters

- **path** (`str`) – The path to a BEL graph in any of the formats with extensions described below
- **kwargs** – The keyword arguments are passed to the importer function

Return type `BELGraph`

Returns A BEL graph.

This is the universal loader, which means any file path can be given and PyBEL will look up the appropriate load function. Allowed extensions are:

- `bel`
- `bel.nodelink.json`
- `bel.cx.json`
- `bel.jgif.json`

The previous extensions also support gzipping. Other allowed extensions that don't support gzip are:

- `bel.pickle` / `bel.gpickle` / `bel.pkl`
- `indra.json`

`pybel.dump(graph, path, **kwargs)`
Write a BEL graph.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **path** (`str`) – The path to which the BEL graph is written.
- **kwargs** – The keyword arguments are passed to the exporter function

This is the universal loader, which means any file path can be given and PyBEL will look up the appropriate writer function. Allowed extensions are:

- `bel`

- `bel.nodelink.json`
- `bel.unodelink.json`
- `bel.cx.json`
- `bel.jgif.json`
- `bel.graphdati.json`

The previous extensions also support gzipping. Other allowed extensions that don't support gzip are:

- `bel.pickle` / `bel.gpickle` / `bel.pkl`
- `indra.json`
- `tsv`
- `gsea`

Return type `None`

19.1 Import

19.1.1 Parsing Modes

The PyBEL parser has several modes that can be enabled and disabled. They are described below.

Allow Naked Names

By default, this is set to `False`. The parser does not allow identifiers that are not qualified with namespaces (*naked names*), like in `p(YFG)`. A proper namespace, like `p(HGNC:YFG)` must be used. By setting this to `True`, the parser becomes permissive to naked names. In general, this is bad practice and this feature will be removed in the future.

Allow Nested

By default, this is set to `False`. The parser does not allow nested statements is disabled. See *overview*. By setting this to `True` the parser will accept nested statements one level deep.

Citation Clearing

By default, this is set to `True`. While the BEL specification clearly states how the language should be used as a state machine, many BEL documents do not conform to the strict SET/UNSET rules. To guard against annotations accidentally carried from one set of statements to the next, the parser has two modes. By default, in citation clearing mode, when a `SET CITATION` command is reached, it will clear all other annotations (except the `STATEMENT_GROUP`, which has higher priority). This behavior can be disabled by setting this to `False` to re-enable strict parsing.

19.1.2 Reference

`pybel.from_bel_script(path, **kwargs)`

Load a BEL graph from a file resource. This function is a thin wrapper around `from_lines()`.

Parameters `path` (`Union[str, TextIO]`) – A path or file-like

The remaining keyword arguments are passed to `pybel.io.line_utils.parse_lines()`, which populates a `BELGraph`.

Return type `BELGraph`

`pybel.from_bel_script_url(url, **kwargs)`

Load a BEL graph from a URL resource.

Parameters `url` (`str`) – A valid URL pointing to a BEL document

The remaining keyword arguments are passed to `pybel.io.line_utils.parse_lines()`.

Return type `BELGraph`

`pybel.to_bel_script(graph, path, use_identifiers=True)`

Write the `BELGraph` as a canonical BEL script.

Parameters

- `graph` (`BELGraph`) – the BEL Graph to output as a BEL Script
- `path` (`Union[str, TextIO]`) – A path or file-like.
- `use_identifiers` (`bool`) – Enables extended [BEP-0008](#) syntax

Return type `None`

19.1.3 Hetionet

Importer for Hetionet JSON.

`pybel.from_hetionet_json(hetionet_dict, use_tqdm=True)`

Convert a Hetionet dictionary to a BEL graph.

Return type `BELGraph`

`pybel.from_hetionet_file(file)`

Get Hetionet from a JSON file.

Return type `BELGraph`

`pybel.from_hetionet_gz(path)`

Get Hetionet from its JSON GZ file.

Return type `BELGraph`

`pybel.get_hetionet()`

Get Hetionet from GitHub, cache, and convert to BEL.

Return type `BELGraph`

19.2 Transport

All transport pairs are reflective and data-preserving.

19.2.1 Bytes

Conversion functions for BEL graphs with bytes and Python pickles.

`pybel.from_bytes(bytes_graph, check_version=True)`

Read a graph from bytes (the result of pickling the graph).

Parameters

- **bytes_graph** (`bytes`) – File or filename to write
- **check_version** (`bool`) – Checks if the graph was produced by this version of PyBEL

Return type `BELGraph`

`pybel.to_bytes(graph, protocol=5)`

Convert a graph to bytes with pickle.

Note that the pickle module has some incompatibilities between Python 2 and 3. To export a universally importable pickle, choose 0, 1, or 2.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **protocol** (`int`) – Pickling protocol to use. Defaults to `HIGHEST_PROTOCOL`.

See also:

<https://docs.python.org/3.6/library/pickle.html#data-stream-format>

Return type `bytes`

`pybel.from_bytes_gz(bytes_graph)`

Read a graph from gzipped bytes (the result of pickling the graph).

Parameters **bytes_graph** (`bytes`) – File or filename to write

Return type `BELGraph`

`pybel.to_bytes_gz(graph, protocol=5)`

Convert a graph to gzipped bytes with pickle.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **protocol** (`int`) – Pickling protocol to use. Defaults to `HIGHEST_PROTOCOL`.

Return type `bytes`

`pybel.from_pickle(path, check_version=True)`

Read a graph from a pickle file.

Parameters

- **path** (`Union[str, BinaryIO]`) – File or filename to read. Filenames ending in `.gz` or `.bz2` will be uncompressed.
- **check_version** (`bool`) – Checks if the graph was produced by this version of PyBEL

Return type *BELGraph*

`pybel.to_pickle(graph, path, protocol=5)`

Write this graph to a pickle file.

Note that the pickle module has some incompatibilities between Python 2 and 3. To export a universally importable pickle, choose 0, 1, or 2.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **path** (`Union[str, BinaryIO]`) – A path or file-like
- **protocol** (`int`) – Pickling protocol to use. Defaults to `HIGHEST_PROTOCOL`.

See also:

<https://docs.python.org/3.6/library/pickle.html#data-stream-format>

Return type `None`

`pybel.from_pickle_gz(path)`

Read a graph from a gzipped pickle file.

Return type *BELGraph*

`pybel.to_pickle_gz(graph, path, protocol=5)`

Write this graph to a gzipped pickle file.

Return type `None`

19.2.2 Node-Link JSON

Conversion functions for BEL graphs with node-link JSON.

`pybel.from_nodelink(graph_json_dict, check_version=True)`

Build a graph from node-link JSON Object.

Return type *BELGraph*

`pybel.to_nodelink(graph)`

Convert this graph to a node-link JSON object.

Parameters **graph** (*BELGraph*) – BEL Graph

Return type `Mapping[str, Any]`

`pybel.from_nodelink_jsons(graph_json_str, check_version=True)`

Read a BEL graph from a node-link JSON string.

Return type *BELGraph*

`pybel.to_nodelink_jsons(graph, **kwargs)`

Dump this graph as a node-link JSON object to a string.

Return type `str`

`pybel.from_nodelink_file(path, check_version=True)`

Build a graph from the node-link JSON contained in the given file.

Parameters **path** (`Union[str, TextIO]`) – A path or file-like

Return type *BELGraph*

`pybel.to_nodelink_file(graph, path, **kwargs)`

Write this graph as node-link JSON to a file.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **path** (*Union[str, TextIO]*) – A path or file-like

Return type *None*

`pybel.from_nodelink_gz(path)`

Read a graph as node-link JSON from a gzip file.

Return type *BELGraph*

`pybel.to_nodelink_gz(graph, path, **kwargs)`

Write a graph as node-link JSON to a gzip file.

Return type *None*

19.2.3 Streamable BEL (JSONL)

Streamable BEL as JSON.

`pybel.from_sbel(it, includes_metadata=True)`

Load a BEL graph from an iterable of dictionaries corresponding to lines in BEL JSONL.

Parameters

- **it** (*Iterable[Any]*) – An iterable of dictionaries.
- **includes_metadata** (*bool*) – By default, interprets the first element of the iterable as the graph's metadata. Switch to *False* to disable.

Return type *BELGraph*

Returns A BEL graph

`pybel.to_sbel(graph)`

Create a list of JSON dictionaries corresponding to lines in BEL JSONL.

Return type *List[Any]*

`pybel.from_sbel_file(path)`

Build a graph from the BEL JSONL contained in the given file.

Parameters **path** (*Union[str, TextIO]*) – A path or file-like

Return type *BELGraph*

`pybel.to_sbel_file(graph, path, separators=(',', ':'), **kwargs)`

Write this graph as BEL JSONL to a file.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **separators** – The separators used in `json.dumps()`
- **path** (*Union[str, TextIO]*) – A path or file-like

Return type *None*

`pybel.from_sbel_gz(path)`

Read a graph as BEL JSONL from a gzip file.

Return type [*BELGraph*](#)

`pybel.to_sbel_gz(graph, path, separators=(',', ':'), **kwargs)`
Write a graph as BEL JSONL to a gzip file.

Parameters

- **graph** ([*BELGraph*](#)) – A BEL graph
- **separators** – The separators used in `json.dumps()`
- **path** ([*str*](#)) – A path for a gzip file

Return type [*None*](#)

19.2.4 Cyberinfrastructure Exchange

This module wraps conversion between [*pybel.BELGraph*](#) and the Cyberinfrastructure Exchange (CX) JSON.

CX is an aspect-oriented network interchange format encoded in JSON with a format inspired by the JSON-LD encoding of Resource Description Framework (RDF). It is primarily used by the Network Data Exchange (NDEx) and more recent versions of Cytoscape.

See also:

- The NDEx Data Model [Specification](#)
- [Cytoscape.js](#)
- CX Support for Cytoscape.js on the Cytoscape [App Store](#)

`pybel.from_cx(cx)`
Rebuild a [*BELGraph*](#) from CX JSON output from PyBEL.

Parameters **cx** ([*List\[Dict\]*](#)) – The CX JSON object for this graph

Return type [*BELGraph*](#)

`pybel.to_cx(graph)`
Convert a BEL Graph to a CX JSON object for use with [NDEx](#).

See also:

- [NDEx Python Client](#)

Return type [*List\[Dict\]*](#)

`pybel.from_cx_jsons(graph_json_str)`
Read a BEL graph from a CX JSON string.

Return type [*BELGraph*](#)

`pybel.to_cx_jsons(graph, **kwargs)`
Dump this graph as a CX JSON object to a string.

Return type [*str*](#)

`pybel.from_cx_file(path)`
Read a file containing CX JSON and converts to a BEL graph.

Parameters **path** ([*Union\[str, TextIO\]*](#)) – A readable file or file-like containing the CX JSON for this graph

Return type [*BELGraph*](#)

Returns A BEL Graph representing the CX graph contained in the file

`pybel.to_cx_file(graph, path, indent=2, **kwargs)`

Write a BEL graph to a JSON file in CX format.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **path** (*Union[str, TextIO]*) – A writable file or file-like
- **indent** (*Optional[int]*) – How many spaces to use to pretty print. Change to None for no pretty printing

The example below shows how to output a BEL graph as CX to an open file.

```
from pybel.examples import sialic_acid_graph
from pybel import to_cx_file
with open('graph.bel.cx.json', 'w') as file:
    to_cx_file(sialic_acid_graph, file)
```

The example below shows how to output a BEL graph as CX to a file at a given path.

```
from pybel.examples import sialic_acid_graph
from pybel import to_cx_file
to_cx_file(sialic_acid_graph, 'graph.bel.cx.json')
```

If you have a big graph, you might consider storing it as a gzipped JGIF file by using `to_cx_gz()`.

Return type *None*

`pybel.from_cx_gz(path)`

Read a graph as CX JSON from a gzip file.

Return type *BELGraph*

`pybel.to_cx_gz(graph, path, **kwargs)`

Write a graph as CX JSON to a gzip file.

Return type *None*

19.2.5 JSON Graph Interchange Format

Conversion functions for BEL graphs with JGIF JSON.

The JSON Graph Interchange Format (JGIF) is [specified](#) similarly to the Node-Link JSON. Interchange with this format provides compatibility with other software and repositories, such as the [Causal Biological Network Database](#).

`pybel.from_jgif(graph_jgif_dict, parser_kwargs=None)`

Build a BEL graph from a JGIF JSON object.

Parameters **graph_jgif_dict** (*dict*) – The JSON object representing the graph in JGIF format

Return type *BELGraph*

`pybel.to_jgif(graph)`

Build a JGIF dictionary from a BEL graph.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A JGIF dictionary

Return type *dict*

Warning: Untested! This format is not general purpose and is therefore time is not heavily invested. If you want to use Cytoscape.js, we suggest using `pybel.to_cx()` instead.

The example below shows how to output a BEL graph as a JGIF dictionary.

```
import os
from pybel.examples import sialic_acid_graph
graph_jgif_json = pybel.to_jgif(sialic_acid_graph)
```

If you want to write the graph directly to a file as JGIF, see func:`to_jgif_file`.

`pybel.from_jgif_jsons(graph_json_str)`

Read a BEL graph from a JGIF JSON string.

Return type `BELGraph`

`pybel.to_jgif_jsons(graph, **kwargs)`

Dump this graph as a JGIF JSON object to a string.

Return type `str`

`pybel.from_jgif_file(path)`

Build a graph from the JGIF JSON contained in the given file.

Parameters `path` (`Union[str, TextIO]`) – A path or file-like

Return type `BELGraph`

`pybel.to_jgif_file(graph, file, **kwargs)`

Write JGIF to a file.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **file** (`Union[str, TextIO]`) – A writable file or file-like

The example below shows how to output a BEL graph as JGIF to an open file.

```
from pybel.examples import sialic_acid_graph
from pybel import to_jgif_file
with open('graph.bel.jgif.json', 'w') as file:
    to_jgif_file(sialic_acid_graph, file)
```

The example below shows how to output a BEL graph as JGIF to a file at a given path.

```
from pybel.examples import sialic_acid_graph
from pybel import to_jgif_file
to_jgif_file(sialic_acid_graph, 'graph.bel.jgif.json')
```

If you have a big graph, you might consider storing it as a gzipped JGIF file by using `to_jgif_gz()`.

Return type `None`

`pybel.from_jgif_gz(path)`

Read a graph as JGIF JSON from a gzip file.

Return type `BELGraph`

`pybel.to_jgif_gz(graph, path, **kwargs)`

Write a graph as JGIF JSON to a gzip file.

Return type `None`

`pybel.post_jgif(graph, url, **kwargs)`
Post the JGIF to a given URL.

Return type `Response`

`pybel.from_cbn_jgif(graph_jgif_dict)`
Build a BEL graph from CBN JGIF.

Map the JGIF used by the Causal Biological Network Database to standard namespace and annotations, then builds a BEL graph using `pybel.from_jgif()`.

Parameters `graph_jgif_dict` (`dict`) – The JSON object representing the graph in JGIF format

Return type `BELGraph`

Example: .. code-block:: python

```
import requests
from pybel import from_cbn_jgif
apoptosis_url = 'http://causalbionet.com/Networks/GetJSONGraphFile?networkId=810385422'
graph_jgif_dict = requests.get(apoptosis_url).json()
graph = from_cbn_jgif(graph_jgif_dict)
```

Warning: Handling the annotations is not yet supported, since the CBN documents do not refer to the resources used to create them. This may be added in the future, but the annotations must be stripped from the graph before uploading to the network store using `pybel.struct.mutation.strip_annotations()`.

`pybel.from_cbn_jgif_file(path)`
Build a graph from a file containing the CBN variant of JGIF.

Parameters `path` (`Union[str, TextIO]`) – A path or file-like

Return type `BELGraph`

19.2.6 GraphDati

Conversion functions for BEL graphs with GraphDati.

Note that these are not exact I/O - you can't currently use them as a round trip because the input functions expect the GraphDati format that's output by BioDati.

`pybel.to_graphdati(graph, *, use_identifiers=True, skip_unqualified=True, use_tqdm=False, metadata_extras=None)`

Export a GraphDati list using the nanopub.

Parameters

- **graph** – A BEL graph
- **use_identifiers** (`bool`) – use OBO-style identifiers
- **use_tqdm** (`bool`) – Show a progress bar while generating nanopubs
- **skip_unqualified** (`bool`) – Should unqualified edges be output as nanopubs? Defaults to false.
- **metadata_extras** (`Optional[Mapping[str, Any]]`) – Extra information to pass into the metadata part of nanopubs

Return type `List[Mapping[str, Mapping[str, Any]]]`

`pybel.from_graphdati(j, use_tqdm=True)`
 Convert data from the “normal” network format.

Warning: BioDati crashes when requesting the full network format, so this isn’t yet explicitly supported

Return type `BELGraph`

`pybel.to_graphdati_file(graph, path, use_identifiers=True, **kwargs)`
 Write this graph as GraphDati JSON to a file.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **path** (`Union[str, TextIO]`) – A path or file-like

Return type `None`

`pybel.from_graphdati_file(path)`
 Load a file containing GraphDati JSON.

Parameters **path** (`Union[str, TextIO]`) – A path or file-like

Return type `BELGraph`

`pybel.to_graphdati_gz(graph, path, **kwargs)`
 Write a graph as GraphDati JSON to a gzip file.

Return type `None`

`pybel.from_graphdati_gz(path)`
 Read a graph as GraphDati JSON from a gzip file.

Return type `BELGraph`

`pybel.to_graphdati_jsons(graph, **kwargs)`
 Dump this graph as a GraphDati JSON object to a string.

Parameters **graph** (`BELGraph`) – A BEL graph

Return type `str`

`pybel.from_graphdati_jsons(s)`
 Load a graph from a GraphDati JSON string.

Parameters **graph** – A BEL graph

Return type `BELGraph`

`pybel.to_graphdati_jsonl(graph, file, use_identifiers=True, use_tqdm=True)`
 Write this graph as a GraphDati JSON lines file.

Parameters **graph** – A BEL graph

`pybel.to_graphdati_jsonl_gz(graph, path, **kwargs)`
 Write a graph as GraphDati JSONL to a gzip file.

Parameters **graph** (`BELGraph`) – A BEL graph

Return type `None`

19.2.7 INDRA

Conversion functions for BEL graphs with INDRA.

After assembling a model with [INDRA](#), a list of `indra.statements.Statement` can be converted to a [pybel.BELGraph](#) with `indra.assemblers.pybel.PybelAssembler`.

```
from indra.assemblers.pybel import PybelAssembler
import pybel

stmts = [
    # A list of INDRA statements
]

pba = PybelAssembler(
    stmts,
    name='Graph Name',
    version='0.0.1',
    description='Graph Description'
)
graph = pba.make_model()

# Write to BEL file
pybel.to_bel_path(belgraph, 'simple_pybel.bel')
```

Warning: These functions are hard to unit test because they rely on a whole set of java dependencies and will likely not be for a while.

`pybel.from_indra_statements(stmts, name=None, version=None, description=None, authors=None, contact=None, license=None, copyright=None, disclaimer=None)`

Import a model from indra.

Parameters

- **stmts** (`List[indra.statements.Statement]`) – A list of statements
- **name** (`Optional[str]`) – The graph's name
- **version** (`Optional[str]`) – The graph's version. Recommended to use semantic versioning or YYYYMMDD format.
- **description** (`Optional[str]`) – The description of the graph
- **authors** (`Optional[str]`) – The authors of this graph
- **contact** (`Optional[str]`) – The contact email for this graph
- **license** (`Optional[str]`) – The license for this graph
- **copyright** (`Optional[str]`) – The copyright for this graph
- **disclaimer** (`Optional[str]`) – The disclaimer for this graph

Return type `pybel.BELGraph`

`pybel.from_indra_statements_json(stmts_json, **kwargs)`

Get a BEL graph from INDRA statements JSON.

Return type `BELGraph`

Other kwargs are passed to `from_indra_statements()`.

`pybel.from_indra_statements_json_file(file, **kwargs)`

Get a BEL graph from INDRA statements JSON file.

Return type `BELGraph`

Other kwargs are passed to `from_indra_statements()`.

`pybel.to_indra_statements(graph)`

Export this graph as a list of INDRA statements using the `indra.sources.pybel.PybelProcessor`.

Parameters `graph` (`pybel.BELGraph`) – A BEL graph

Return type `list[indra.statements.Statement]`

`pybel.to_indra_statements_json(graph)`

Export this graph as INDRA JSON list.

Parameters `graph` (`pybel.BELGraph`) – A BEL graph

Return type `List[Mapping[str, Any]]`

`pybel.to_indra_statements_json_file(graph, path, indent=2, **kwargs)`

Export this graph as INDRA statement JSON.

Parameters

- `graph` (`pybel.BELGraph`) – A BEL graph
- `path` (`Union[str, TextIO]`) – A writable file or file-like

Other kwargs are passed to `json.dump()`.

`pybel.from_biopax(path, encoding=None, **kwargs)`

Import a model encoded in Pathway Commons BioPAX via `indra`.

Parameters

- `path` (`str`) – Path to a BioPAX OWL file
- `encoding` (`Optional[str]`) – The encoding passed to `indra.sources.biopax.process_owl()`. See <https://github.com/sorgerlab/indra/pull/1199>.

Return type `pybel.BELGraph`

Other kwargs are passed to `from_indra_statements()`.

Warning: Not compatible with all BioPAX! See INDRA documentation.

19.3 Visualization

19.3.1 Jupyter

Support for displaying BEL graphs in Jupyter notebooks.

`pybel.to_jupyter(graph, width=1000, height=650, color_map=None)`

Display a BEL graph inline in a Jupyter notebook.

To use successfully, make run as the last statement in a cell inside a Jupyter notebook.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **width** (*int*) – The width of the visualization window to render
- **height** (*int*) – The height of the visualization window to render
- **color_map** (*Optional[Mapping[str, str]]*) – A dictionary from PyBEL internal node functions to CSS color strings like #FFEE00. Defaults to `default_color_map`

Returns An IPython notebook Javascript object

Return type `IPython.display.Javascript`

19.4 Analytical Services

19.4.1 PyNPA

Exporter for PyNPA.

See also:

<https://github.com/pynpa>

`pybel.to_npa_directory(graph, directory, **kwargs)`

Write the BEL file to two files in the directory for pynpa.

Return type `None`

`pybel.to_npa_dfs(graph, cartesian_expansion=False, nomenclature_method_first_layer=None, nomenclature_method_second_layer=None, direct_tf_only=False)`

Export the BEL graph as two lists of triples for the pynpa.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **cartesian_expansion** (*bool*) – If true, applies cartesian expansion on both reactions (reactants x products) as well as list abundances using `list_abundance_cartesian_expansion()` and `reaction_cartesian_expansion()`
- **nomenclature_method_first_layer** (*Optional[str]*) – Either “curie”, “name” or “inodes. Defaults to “curie”.
- **nomenclature_method_second_layer** (*Optional[str]*) – Either “curie”, “name” or “inodes. Defaults to “curie”.

1. Pick out all transcription factor relationships. Protein X is a transcription factor for gene Y IFF `complex(p(X), g(Y)) -> r(Y)`
2. Get all other interactions between any gene/rna/protein that are directed causal for the PPI layer

Return type `Tuple[DataFrame, DataFrame]`

19.4.2 HiPathia

Convert a BEL graph to HiPathia inputs.

Input

SIF File

- Text file with three columns separated by tabs.
- Each row represents an interaction in the pathway. First column is the source node, third column the target node, and the second is the type of relation between them.
- Only activation and inhibition interactions are allowed.
- The name of the nodes in this file will be stored as the IDs of the nodes.
- The nodes IDs should have the following structure: N (dash) pathway ID (dash) node ID.
- HiPathia distinguish between two types of nodes: simple and complex.

Simple nodes:

- Simple nodes may include many genes, but only one is needed to perform the function of the node. This could correspond to a protein family of enzymes that all have the same function - only one of them needs to be present for the action to take place. Simple nodes are defined within
- Node IDs from simple nodes do not include any space, i.e. N-hsa04370-11.

Complex nodes:

- Complex nodes include different simple nodes and represent protein complexes. Each simple node within the complex represents one protein in the complex. This node requires the presence of all their simple nodes to perform its function.
- Node IDs from complex nodes are the juxtaposition of the included simple node IDs, separated by spaces, i.e. N-hsa04370-10 26.

ATT File

Text file with twelve (12) columns separated by tabulars. Each row represents a node (either simple or complex).

The columns included are:

1. ID: Node ID as explained above.
2. label: Name to be shown in the picture of the pathway on HGNC. Generally, the gene name of the first included EntrezID gene is used as label. For complex nodes, we juxtapose the gene names of the first genes of each simple node included (see genesList column below).
3. X: The X-coordinate of the position of the node in the pathway.
4. Y: The Y-coordinate of the position of the node in the pathway.
5. color: The default color of the node.
6. shape: The shape of the node. "rectangle" should be used for genes and "circle" for metabolites.
7. type: The type of the node, either "gene" for genes or "compound" for metabolites. For complex nodes, the type of each of their included simple nodes is juxtaposed separated by commas, i.e. gene, gene.
8. label.cex: Amount by which plotting label should be scaled relative to the default.

9. `label.color`: Default color of the node.
10. `width`: Default width of the node.
11. `height`: Default height of the node.
12. `genesList`: List of genes included in each node, with EntrezID:
 - Simple nodes: EntrezIDs of the genes included, separated by commas (“,”) and no spaces, i.e. 56848,8877 for node N-hsa04370-11.
 - Complex nodes: GenesList of the simple nodes included, separated by a slash (“/”) and no spaces, and in the same order as in the node ID. For example, node N-hsa04370-10 26 includes two simple nodes: 10 and 26. Its `genesList` column is 5335,5336,/9047, meaning that the genes included in node 10 are 5335 and 5336, and the gene included in node 26 is 9047.

`pybel.to_hipathia(graph, directory, draw=True)`
Export HiPathia artifacts for the graph.

Return type `None`

`pybel.to_hipathia_dfs(graph, draw_directory=None)`
Get the ATT and SIF dataframes.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **draw_directory** (*Optional[str]*) – The directory in which a drawing should be output

1. Identify nodes: 1. Identify all proteins 2. Identify all protein families 3. Identify all complexes with just a protein or a protein family in them
2. Identify interactions between any of those things that are causal
3. Profit!

Return type `Union[Tuple[None, None], Tuple[DataFrame, DataFrame]]`

`pybel.from_hipathia_paths(name, att_path, sif_path)`
Get a BEL graph from HiPathia files.

Return type *BELGraph*

`pybel.from_hipathia_dfs(name, att_df, sif_df)`
Get a BEL graph from HiPathia dataframes.

Return type *BELGraph*

19.4.3 SPIA

An exporter for signaling pathway impact analysis (SPIA) described by [Tarca2009].

See also:

<https://bioconductor.org/packages/release/bioc/html/SPIA.html>

`pybel.to_spia_dfs(graph)`
Create an excel sheet ready to be used in SPIA software.

Parameters **graph** (*BELGraph*) – BELGraph

Return type `Mapping[str, DataFrame]`

Returns dictionary with matrices

`pybel.to_spia_excel(graph, path)`

Write the BEL graph as an SPIA-formatted excel sheet at the given path.

Return type `None`

`pybel.to_spia_tsvs(graph, directory)`

Write the BEL graph as a set of SPIA-formatted TSV files in a given directory.

Return type `None`

19.4.4 PyKEEN

Entry points for PyKEEN.

PyKEEN is a machine learning library for knowledge graph embeddings that supports node clustering, link prediction, entity disambiguation, question/answering, and other tasks with knowledge graphs. It provides an interface for registering plugins using Python's entrypoints under the `pykeen.triples.extension_importer` and `pykeen.triples.prefix_importer` groups. More specific information about how the PyBEL plugins are loaded into PyKEEN can be found in PyBEL's `setup.cfg` under the `[options.entry_points]` header.

The following example shows how you can parse/load the triples from a BEL document with the `*.bel` extension.

```
from urllib.request import urlretrieve
url = 'https://raw.githubusercontent.com/cthoyt/selventa-knowledge/master/selventa_
↳knowledge/small_corpus.bel'
urlretrieve(url, 'small_corpus.bel')

# Example 1A: Make triples factory
from pykeen.triples import TriplesFactory
tf = TriplesFactory(path='small_corpus.bel')

# Example 1B: Use directly in the pipeline, which automatically invokes training/testing_
↳set stratification
from pykeen.pipeline import pipeline
results = pipeline(
    dataset='small_corpus.bel',
    model='TransE',
)
```

The same is true for precompiled BEL documents in the node-link format with the `*.bel.nodelink.json` extension and the pickle format with the `*.bel.pickle` extension.

The following example shows how you can load/parse the triples from a BEL document stored in BEL Commons using the `bel-commons` prefix in combination with the network's identifier.

```
# Example 2A: Make a triples factory
from pykeen.triples import TriplesFactory
# the network's identifier is 528
tf = TriplesFactory(path='bel-commons:528')

# Example 1B: Use directly in the pipeline, which automatically invokes training/testing_
↳set stratification
from pykeen.pipeline import pipeline
results = pipeline(
```

(continues on next page)

(continued from previous page)

```
dataset='bel-commons:528',
model='TransR',
)
```

Currently, this relies on the default BEL Commons service provider at <https://bel-commons-dev.scai.fraunhofer.de>, whose location might change in the future.

`pybel.io.pykeen.get_triples_from_bel(path)`

Get triples from a BEL file by wrapping `pybel.io.tsv.api.get_triples()`.

Parameters `path` (`str`) – the file path to a BEL Script

Return type `ndarray`

Returns A three column array with head, relation, and tail in each row

`pybel.io.pykeen.get_triples_from_bel_nodelink(path)`

Get triples from a BEL Node-link JSON file by wrapping `pybel.io.tsv.api.get_triples()`.

Parameters `path` (`str`) – the file path to a BEL Node-link JSON file

Return type `ndarray`

Returns A three column array with head, relation, and tail in each row

`pybel.io.pykeen.get_triples_from_bel_pickle(path)`

Get triples from a BEL pickle file by wrapping `pybel.io.tsv.api.get_triples()`.

Parameters `path` (`str`) – the file path to a BEL pickle file

Return type `ndarray`

Returns A three column array with head, relation, and tail in each row

`pybel.io.pykeen.get_triples_from_bel_commons(network_id)`

Load a BEL document from BEL Commons by wrapping `pybel.io.tsv.api.get_triples()`.

Parameters `network_id` (`str`) – The network identifier for a graph in BEL Commons

Return type `ndarray`

Returns A three column array with head, relation, and tail in each row

19.4.5 Machine Learning

Export functions for Machine Learning.

While BEL is a fantastic medium for storing metadata and high granularity information on edges, machine learning algorithms can not consume BEL graphs directly. This module provides functions that make inferences and interpretations of BEL graphs in order to interface with machine learning platforms. One example where we've done this is [BioKEEN](#), which uses this module to convert BEL graphs into a format for knowledge graph embeddings.

`pybel.to_triples(graph, use_tqdm=False, raise_on_none=False)`

Get a non-redundant list of triples representing the graph.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **use_tqdm** (`bool`) – Should a progress bar be shown?
- **raise_on_none** (`bool`) – Should an exception be raised if no triples are returned?

Raises NoTriplesValueError

Return type List[Tuple[str, str, str]]

`pybel.to_triples_file(graph, path, *, use_tqdm=False, sep='\t', raise_on_none=False)`

Write the graph as a TSV.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **path** (*Union[str, TextIO]*) – A path or file-like
- **use_tqdm** (*bool*) – Should a progress bar be shown?
- **sep** – The separator to use
- **raise_on_none** (*bool*) – Should an exception be raised if no triples are returned?

Raises NoTriplesValueError

Return type None

`pybel.to_edgelist(graph, path, *, use_tqdm=False, sep='\t', raise_on_none=False)`

Write the graph as an edgelist.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **path** (*Union[str, TextIO]*) – A path or file-like
- **use_tqdm** (*bool*) – Should a progress bar be shown?
- **sep** – The separator to use
- **raise_on_none** (*bool*) – Should an exception be raised if no triples are returned?

Raises NoTriplesValueError

Return type None

19.5 Web Services

19.5.1 BEL Commons

Transport functions for *BEL Commons*.

BEL Commons is a free, open-source platform for hosting BEL content. Because it was originally developed and published in an academic capacity at Fraunhofer SCAI, a public instance can be found at <https://bel-commons-dev.scai.fraunhofer.de>. However, this instance is only supported out of posterity and will not be updated. If you would like to host your own instance of BEL Commons, there are instructions on its GitHub page.

`pybel.from_bel_commons(network_id, host=None)`

Retrieve a public network from BEL Commons.

In the future, this function may be extended to support authentication.

Parameters

- **network_id** (*int*) – The BEL Commons network identifier
- **host** (*Optional[str]*) – The location of the BEL Commons server. Alternatively, looks up in PyBEL config with PYBEL_REMOTE_HOST or the environment as PYBEL_REMOTE_HOST.

Raises `ValueError` if host configuration can not be found

Return type `BELGraph`

`pybel.to_bel_commons(graph, host=None, user=None, password=None, public=True)`

Send a graph to the receiver service and returns the `requests` response object.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **host** (`Optional[str]`) – The location of the BEL Commons server. Alternatively, looks up in PyBEL config with `PYBEL_REMOTE_HOST` or the environment as `PYBEL_REMOTE_HOST`.
- **user** (`Optional[str]`) – Username for BEL Commons. Alternatively, looks up in PyBEL config with `PYBEL_REMOTE_USER` or the environment as `PYBEL_REMOTE_USER`
- **password** (`Optional[str]`) – Password for BEL Commons. Alternatively, looks up in PyBEL config with `PYBEL_REMOTE_PASSWORD` or the environment as `PYBEL_REMOTE_PASSWORD`
- **public** (`bool`) – Should the network be made public?

Return type `Response`

Returns The response object from `requests`

19.5.2 Amazon Simple Storage Service (S3)

Transport functions for Amazon Web Services (AWS).

AWS has a cloud-based file storage service called S3 that can be programmatically accessed using the `boto3` package. This module provides functions for quickly wrapping upload/download of BEL graphs using the gzipped Node-Link schema.

`pybel.to_s3(graph, *, bucket, key, client=None)`

Save BEL to S3 as gzipped node-link JSON.

If you don't specify an instantiated client, PyBEL will do its best to load a default one using `boto3.client()` like in the following example:

```
import pybel
from pybel.examples import sialic_acid_graph

graph = pybel.to_s3(
    sialic_acid_graph,
    bucket='your bucket',
    key='your file name.bel.nodelink.json.gz',
)
```

However, if you would like to configure your own, you can do it with something like this:

```
import boto3
s3_client = boto3.client('s3')

import pybel
from pybel.examples import sialic_acid_graph

graph = pybel.to_s3(
```

(continues on next page)

(continued from previous page)

```

sialic_acid_graph,
client=s3_client,
bucket='your bucket',
key='your file name.bel.nodelink.json.gz',
)

```

Warning: This assumes you already have credentials set up on your machine

If you don't already have a bucket, you can create one using boto3 by following this tutorial: <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-example-creating-buckets.html>

Return type `None`

`pybel.from_s3(*, bucket, key, client=None)`

Get BEL from gzipped node-link JSON from Amazon S3.

If you don't specify an instantiated client, PyBEL will do its best to load a default one using `boto3.client()` like in the following example:

```

graph = pybel.from_s3(bucket='your bucket', key='your file name.bel.nodelink.json.gz
→')

```

However, if you would like to configure your own, you can do it with something like this:

```

import boto3
s3_client = boto3.client('s3')

import pybel
graph = pybel.from_s3(
    client=s3_client,
    bucket='your bucket',
    key='your file name.bel.nodelink.json.gz',
)

```

Return type `BELGraph`

19.5.3 BioDati

Transport functions for `BioDati`.

BioDati is a paid, closed-source platform for hosting BEL content. However, they do have a demo instance running at <https://studio.demo.biodati.com> with which the examples in this module will be described.

As noted in the transport functions for BioDati, you should change the URLs to point to your own instance of BioDati. If you're looking for an open source storage system for hosting your own BEL content, you may consider [BEL Commons](#), with the caveat that it is currently maintained in an academic capacity. Disclosure: BEL Commons is developed by the developers of PyBEL.

```

pybel.to_biodati(graph, *, username='demo@biodati.com', password='demo',
                 base_url='https://nanopubstore.demo.biodati.com', chunksize=None, use_tqdm=True,
                 collections=None, overwrite=False, validate=True, email=False)

```

Post this graph to a BioDati server.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **username** (*str*) – The email address to log in to BioDati. Defaults to “demo@biodati.com” for the demo server
- **password** (*str*) – The password to log in to BioDati. Defaults to “demo” for the demo server
- **base_url** (*str*) – The BioDati nanopub store base url. Defaults to “https://nanopubstore.demo.biodati.com” for the demo server’s nanopub store
- **chunksize** (*Optional[int]*) – The number of nanopubs to post at a time. By default, does all.
- **use_tqdm** (*bool*) – Should tqdm be used when iterating?
- **collections** (*Optional[Iterable[str]]*) – Tags to add to the nanopubs for lookup on BioDati
- **overwrite** (*bool*) – Set the BioDati upload “overwrite” setting
- **validate** (*bool*) – Set the BioDati upload “validate” setting
- **email** (*Union[bool, str]*) – Who should get emailed with results about the upload? If true, emails to user used for login. If string, emails to that user. If false, no email.

Return type Response

Returns The response from the BioDati server (last response if using chunking)

Warning: BioDati does not support large uploads (yet?).

Warning: The default public BioDati server has been put here. You should switch it to yours. It will look like https://nanopubstore.<YOUR NAME>.biodati.com.

```
pybel.from_biodati(network_id, username='demo@biodati.com', password='demo',
                   base_url='https://networkstore.demo.biodati.com')
```

Get a graph from a BioDati network store based on its network identifier.

Parameters

- **network_id** (*str*) – The internal identifier of the network you want to download.
- **username** (*str*) – The email address to log in to BioDati. Defaults to “demo@biodati.com” for the demo server
- **password** (*str*) – The password to log in to BioDati. Defaults to “demo” for the demo server
- **base_url** (*str*) – The BioDati network store base url. Defaults to “https://networkstore.demo.biodati.com” for the demo server’s network store

Example usage:

```
from pybel import from_biodati
network_id = '01E46GDFQAGK5W8EFS9S9WMH12' # COVID-19 graph example from Wendy_
↳ Zimmermann
```

(continues on next page)

(continued from previous page)

```
graph = from_biodati(
    network_id=network_id,
    username='demo@biodati.com',
    password='demo',
    base_url='https://networkstore.demo.biodati.com',
)
graph.summarize()
```

Warning: The default public BioDati server has been put here. You should switch it to yours. It will look like `https://networkstore.<YOUR NAME>.biodati.com`.

Return type *BELGraph*

19.5.4 Fraunhofer OrientDB

Transport functions for Fraunhofer's OrientDB.

Fraunhofer hosts an instance of OrientDB that contains BEL in a schema similar to *pybel.io.umbrella_nodelink*. However, they include custom relations that do not come from a controlled vocabulary, and have not made the schema, ETL scripts, or documentation available.

Unlike BioDati and BEL Commons, the Fraunhofer OrientDB does not allow for uploads, so only a single function *pybel.from_fraunhofer_orientdb()* is provided by PyBEL.

pybel.from_fraunhofer_orientdb(database='covid', user='covid_user', password='covid', query=None)

Get a BEL graph from the Fraunhofer OrientDB.

Parameters

- **database** (*str*) – The OrientDB database to connect to
- **user** (*str*) – The user to connect to OrientDB
- **password** (*str*) – The password to connect to OrientDB
- **query** (*Optional[str]*) – The query to run. Defaults to the URL encoded version of `select from E`, where E is all edges in the OrientDB edge database. Likely does not need to be changed, except in the case of selecting specific subsets of edges. Make sure you URL encode it properly, because OrientDB's RESTful API puts it in the URL's path.

By default, this function connects to the `covid` database, that corresponds to the COVID-19 Knowledge Graph⁰. If other databases in the Fraunhofer OrientDB are published and demo username/password combinations are given, the following table will be updated.

Database	Username	Password
covid	covid_user	covid

The `covid` database can be downloaded and converted to a BEL graph like this:

⁰ Domingo-Fernández, D., *et al.* (2020). COVID-19 Knowledge Graph: a computable, multi-modal, cause-and-effect knowledge model of COVID-19 pathophysiology. *bioRxiv* 2020.04.14.040667.

```
import pybel
graph = pybel.from_fraunhofer_orientdb(
    database='covid',
    user='covid_user',
    password='covid',
)
graph.summarize()
```

However, because the source BEL scripts for the COVID-19 Knowledge Graph are available on [GitHub](https://github.com/covid19kg/covid19kg) and the authors pre-enabled it for PyBEL, it can be downloaded with `pip install git+https://github.com/covid19kg/covid19kg.git` and used with the following python code:

```
import covid19kg
graph = covid19kg.get_graph()
graph.summarize()
```

Warning: It was initially planned to handle some of the non-standard relationships listed in the Fraunhofer OrientDB's [schema](#) in their OrientDB Studio instance, but none of them actually appear in the only network that is accessible. If this changes, please leave an issue at <https://github.com/pybel/pybel/issues> so it can be addressed.

Return type *BELGraph*

19.5.5 EMMAA

Ecosystem of Machine-maintained Models with Automated Analysis (EMMAA).

[EMMAA](#) is a project built on top of INDRA by the Sorger Lab at Harvard Medical School. It automatically builds knowledge graphs around pathways/indications periodically (almost daily) using the INDRA Database, which in turn is updated periodically (almost daily) with the most recent literature from MEDLINE, PubMed Central, several major publishers, and other bespoke text corpora such as CORD-19.

`pybel.from_emmaa(model, *, date=None, extension=None, suppress_warnings=False)`

Get an EMMAA model as a BEL graph.

Get the most recent COVID-19 model from EMMAA with the following:

```
import pybel

covid19_emmaa_graph = pybel.from_emmaa('covid19', extension='jsonl')
covid19_emmaa_graph.summarize()
```

PyBEL does its best to look up the most recent model, but if that doesn't work, you can specify it explicitly with the `date` keyword argument in the form of `%Y-%m-%d-%H-%M-%S` like in the following:

```
import pybel

covid19_emmaa_graph = pybel.from_emmaa('covid19', '2020-04-23-17-44-57', extension=
↪ 'jsonl')
covid19_emmaa_graph.summarize()
```

Return type *BELGraph*

19.6 Databases

19.6.1 SQL Databases

Conversion functions for BEL graphs with a SQL database.

`pybel.from_database(name, version=None, manager=None)`

Load a BEL graph from a database.

If `name` and `version` are given, finds it exactly with `pybel.manager.Manager.get_network_by_name_version()`. If just the name is given, finds most recent with `pybel.manager.Manager.get_network_by_name_version()`

Parameters

- **name** (`str`) – The name of the graph
- **version** (`Optional[str]`) – The version string of the graph. If not specified, loads most recent graph added with this name

Returns A BEL graph loaded from the database

Return type `Optional[BELGraph]`

`pybel.to_database(graph, manager=None, use_tqdm=True)`

Store a graph in a database.

Parameters **graph** (`BELGraph`) – A BEL graph

Returns If successful, returns the network object from the database.

Return type `Optional[Network]`

19.6.2 Neo4j

Output functions for BEL graphs to Neo4j.

`pybel.to_neo4j(graph, neo_connection, use_tqdm=False)`

Upload a BEL graph to a Neo4j graph database using `py2neo`.

Parameters

- **graph** (`pybel.BELGraph`) – A BEL Graph
- **neo_connection** (`str` or `py2neo.Graph`) – A `py2neo` connection object. Refer to the [py2neo documentation](#) for how to build this object.

Example Usage:

```
>>> import py2neo
>>> import pybel
>>> from pybel.examples import sialic_acid_graph
>>> neo_graph = py2neo.Graph("http://localhost:7474/db/data/") # use your own
↳connection settings
>>> pybel.to_neo4j(sialic_acid_graph, neo_graph)
```

19.7 Lossy Export

19.7.1 Umbrella Node-Link JSON

The Umbrella Node-Link JSON format is similar to node-link but uses full BEL terms as nodes.

Given a BEL statement describing that *X* phosphorylates *Y* like `act(p(X)) -> p(Y, pmod(Ph))`, PyBEL usually stores the `act()` information about *X* as part of the relationship. In Umbrella mode, this stays as part of the node.

Note that this generates additional nodes in the network for each of the “modified” versions of the node. For example, `act(p(X))` will be represented as individual node instead of `p(X)`, as in the standard node-link JSON exporter.

A user might want to use this exporter in the following scenarios:

- Represent transitivity in activities like in `p(X, pmod(Ph)) -> act(p(X)) -> p(Y, pmod(Ph)) -> act(p(Y))` with four nodes that are more ammenable to simulatons (e.g., boolean networks, petri nets).
- Visualizing networks that in similar way to the legacy BEL [Cytoscape plugin](#) from the BEL Framework (warning: now defunct) using tools like Cytoscape.

`pybel.to_umbrella_nodelink(graph)`

Convert this graph to an umbrella node-link JSON object.

Parameters `graph` (*BELGraph*) – A BEL graph

Return type `Mapping[str, Any]`

`pybel.to_umbrella_nodelink_file(graph, path, **kwargs)`

Write this graph as an umbrella node-link JSON to a file.

Parameters

- `graph` (*BELGraph*) – A BEL graph
- `path` (`Union[str, TextIO]`) – A path or file-like

Return type `None`

`pybel.to_umbrella_nodelink_gz(graph, path, **kwargs)`

Write this graph as an umbrella node-link JSON to a gzipped file.

Return type `None`

19.7.2 GraphML

Conversion functions for BEL graphs with [GraphML](#).

`pybel.to_graphml(graph, path, schema=None)`

Write a graph to a GraphML XML file using `networkx.write_graphml()`.

Parameters

- `graph` (*BELGraph*) – BEL Graph
- `path` (`Union[str, BinaryIO]`) – Path to the new exported file
- `schema` (`Optional[str]`) – Type of export. Currently supported: “simple” and “umbrella”.

The `.graphml` file extension is suggested so Cytoscape can recognize it. By default, this function exports using the PyBEL schema of including modifier information into the edges. As an alternative, this function can also distinguish between

Return type `None`

19.7.3 Miscellaneous

This module contains IO functions for outputting BEL graphs to lossy formats, such as GraphML and CSV.

`pybel.to_csv(graph, path, sep=None)`

Write the graph as a tab-separated edge list.

The resulting file will contain the following columns:

1. Source BEL term
2. Relation
3. Target BEL term
4. Edge data dictionary

See the Data Models section of the documentation for which data are stored in the edge data dictionary, such as queryable information about transforms on the subject and object and their associated metadata.

Return type `None`

`pybel.to_sif(graph, path, sep=None)`

Write the graph as a tab-separated SIF file.

The resulting file will contain the following columns:

1. Source BEL term
2. Relation
3. Target BEL term

This format is simple and can be used readily with many applications, but is lossy in that it does not include relation metadata.

Return type `None`

`pybel.to_gsea(graph, path)`

Write the genes/gene products to a GRP file for use with GSEA gene set enrichment analysis.

See also:

- [GRP format specification](#)
- [GSEA publication](#)

Return type `None`

MANAGER

20.1 Manager API

The BaseManager takes care of building and maintaining the connection to the database via SQLAlchemy.

class `pybel.manager.BaseManager(engine, session)`
A wrapper around a SQLAlchemy engine and session.

Instantiate a manager from an engine and session.

base
alias of `sqlalchemy.orm.decl_api.Base`

create_all(checkfirst=True)
Create the PyBEL cache's database and tables.

Parameters `checkfirst` (`bool`) – Check if the database exists before trying to re-make it

Return type `None`

drop_all(checkfirst=True)
Drop all data, tables, and databases for the PyBEL cache.

Parameters `checkfirst` (`bool`) – Check if the database exists before trying to drop it

Return type `None`

bind()
Bind the metadata to the engine and session.

Return type `None`

The Manager collates multiple groups of functions for interacting with the database. For sake of code clarity, they are separated across multiple classes that are documented below.

class `pybel.manager.Manager(connection=None, engine=None, session=None, **kwargs)`
Bases: `pybel.manager.cache_manager._Manager`

A manager for the PyBEL database.

Create a connection to database and a persistent session using SQLAlchemy.

A custom default can be set as an environment variable with the name `pybel.constants.PYBEL_CONNECTION`, using an [RFC-1738](#) string. For example, a MySQL string can be given with the following form:

`mysql+pymysql://<username>:<password>@<host>/<dbname>?charset=utf8[&<options>]`

A SQLite connection string can be given in the form:

`sqlite:///~/Desktop/cache.db`

Further options and examples can be found on the SQLAlchemy documentation on [engine configuration](#).

Parameters

- **connection** (*Optional[str]*) – An RFC-1738 database connection string. If `None`, tries to load from the environment variable `PYBEL_CONNECTION` then from the config file `~/.config/pybel/config.json` whose value for `PYBEL_CONNECTION` defaults to `pybel.constants.DEFAULT_CACHE_CONNECTION`.
- **engine** – Optional engine to use. Must be specified with a session and no connection.
- **session** – Optional session to use. Must be specified with an engine and no connection.
- **echo** (*bool*) – Turn on echoing sql
- **autoflush** (*Optional[bool]*) – Defaults to `True` if not specified in kwargs or configuration.
- **autocommit** (*Optional[bool]*) – Defaults to `False` if not specified in kwargs or configuration.
- **expire_on_commit** (*Optional[bool]*) – Defaults to `False` if not specified in kwargs or configuration.
- **scopefunc** – Scoped function to pass to `sqlalchemy.orm.scoped_session()`

From the Flask-SQLAlchemy documentation:

An extra key 'scopefunc' can be set on the `options` dict to specify a custom scope function. If it's not provided, Flask's app context stack identity is used. This will ensure that sessions are created and removed with the request/response cycle, and should be fine in most cases.

Allowed Usages:

Instantiation with connection string as positional argument

```
>>> my_connection = 'sqlite:///~/Desktop/cache.db'
>>> manager = Manager(my_connection)
```

Instantiation with connection string as positional argument with keyword arguments

```
>>> my_connection = 'sqlite:///~/Desktop/cache.db'
>>> manager = Manager(my_connection, echo=True)
```

Instantiation with connection string as keyword argument

```
>>> my_connection = 'sqlite:///~/Desktop/cache.db'
>>> manager = Manager(connection=my_connection)
```

Instantiation with connection string as keyword argument with keyword arguments

```
>>> my_connection = 'sqlite:///~/Desktop/cache.db'
>>> manager = Manager(connection=my_connection, echo=True)
```

Instantiation with user-supplied engine and session objects as keyword arguments

```
>>> my_engine, my_session = ... # magical creation! See SQLAlchemy documentation
>>> manager = Manager(engine=my_engine, session=my_session)
```

20.2 Manager Components

class `pybel.manager.NetworkManager(engine, session)`

Groups functions for inserting and querying networks in the database's network store.

Instantiate a manager from an engine and session.

count_networks()

Count the networks in the database.

Return type `int`

list_networks()

List all networks in the database.

Return type `List[Network]`

list_recent_networks()

List the most recently created version of each network (by name).

Return type `List[Network]`

has_name_version(name, version)

Check if there exists a network with the name/version combination in the database.

Return type `bool`

drop_networks()

Drop all networks.

Return type `None`

drop_network_by_id(network_id)

Drop a network by its database identifier.

Return type `None`

drop_network(network)

Drop a network, while also cleaning up any edges that are no longer part of any network.

Return type `None`

query_singleton_edges_from_network(network)

Return a query selecting all edge ids that only belong to the given network.

Return type `Query`

get_network_versions(name)

Return all of the versions of a network with the given name.

Return type `Set[str]`

get_network_by_name_version(name, version)

Load the network with the given name and version if it exists.

Return type `Optional[Network]`

get_graph_by_name_version(name, version)

Load the BEL graph with the given name, or allows for specification of version.

Return type `Optional[BELGraph]`

get_networks_by_name(name)

Get all networks with the given name. Useful for getting all versions of a given network.

Return type `List[Network]`

get_most_recent_network_by_name(*name*)

Get the most recently created network with the given name.

Return type `Optional[Network]`

get_graph_by_most_recent(*name*)

Get the most recently created network with the given name as a `pybel.BELGraph`.

Return type `Optional[BELGraph]`

get_network_by_id(*network_id*)

Get a network from the database by its identifier.

Return type `Network`

get_graph_by_id(*network_id*)

Get a network from the database by its identifier and converts it to a BEL graph.

Return type `BELGraph`

get_networks_by_ids(*network_ids*)

Get a list of networks with the given identifiers.

Note: order is not necessarily preserved.

Return type `List[Network]`

get_graphs_by_ids(*network_ids*)

Get a list of networks with the given identifiers and converts to BEL graphs.

Return type `List[BELGraph]`

get_graph_by_ids(*network_ids*)

Get a combine BEL Graph from a list of network identifiers.

Return type `BELGraph`

class `pybel.manager.QueryManager`(*engine*, *session*)

An extension to the Manager to make queries over the database.

Instantiate a manager from an engine and session.

count_nodes()

Count the number of nodes in the database.

Return type `int`

query_nodes(*bel=None*, *type=None*, *namespace=None*, *name=None*)

Query nodes in the database.

Parameters

- **bel** (`Optional[str]`) – BEL term that describes the biological entity. e.g. `p(HGNC:APP)`
- **type** (`Optional[str]`) – Type of the biological entity. e.g. `Protein`
- **namespace** (`Optional[str]`) – Namespace keyword that is used in BEL. e.g. `HGNC`
- **name** (`Optional[str]`) – Name of the biological entity. e.g. `APP`

Return type `List[Node]`

count_edges()

Count the number of edges in the database.

Return type `int`

get_edges_with_citation(*citation*)

Get the edges with the given citation.

Return type `List[Edge]`

get_edges_with_citations(*citations*)

Get edges with one of the given citations.

Return type `List[Edge]`

search_edges_with_evidence(*evidence*)

Search edges with the given evidence.

Parameters **evidence** (`str`) – A string to search evidences. Can use wildcard percent symbol (%).

Return type `List[Edge]`

search_edges_with_bel(*bel*)

Search edges with given BEL.

Parameters **bel** (`str`) – A BEL string to use as a search

Return type `List[Edge]`

get_edges_with_annotation(*annotation, value*)

Search edges with the given annotation/value pair.

Return type `List[Edge]`

query_edges(*bel=None, source_function=None, source=None, target_function=None, target=None, relation=None*)

Return a query over the edges in the database.

Usually this means that you should call `list()` or `.all()` on this result.

Parameters

- **bel** (`Optional[str]`) – BEL statement that represents the desired edge.
- **source_function** (`Optional[str]`) – Filter source nodes with the given BEL function
- **source** (`Union[None, str, Node]`) – BEL term of source node e.g. `p(HGNC:APP)` or `Node` object.
- **target_function** (`Optional[str]`) – Filter target nodes with the given BEL function
- **target** (`Union[None, str, Node]`) – BEL term of target node e.g. `p(HGNC:APP)` or `Node` object.
- **relation** (`Optional[str]`) – The relation that should be present between source and target node.

query_citations(*db=None, db_id=None, name=None, author=None, date=None, evidence_text=None*)

Query citations in the database.

Parameters

- **db** (`Optional[str]`) – Type of the citation. e.g. `PubMed`
- **db_id** (`Optional[str]`) – The identifier used for the citation. e.g. `PubMed_ID`
- **name** (`Optional[str]`) – Title of the citation.
- **author** (`Union[None, str, List[str]]`) – The name or a list of names of authors participated in the citation.

- **date** (`Union[None, str, date]`) – Publishing date of the citation.
- **evidence_text** (`Optional[str]`) –

Return type `List[Citation]`

query_edges_by_pubmed_identifiers(*pubmed_identifiers*)

Get all edges annotated to the documents identified by the given PubMed identifiers.

Return type `List[Edge]`

query_induction(*nodes*)

Get all edges between any of the given nodes (minimum length of 2).

Return type `List[Edge]`

query_neighbors(*nodes*)

Get all edges incident to any of the given nodes.

Return type `List[Edge]`

MODELS

This module contains the SQLAlchemy database models that support the definition cache and graph cache.

class `pybel.manager.models.Base(**kwargs)`

The most base type

A simple constructor that allows initialization from kwargs.

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

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

class `pybel.manager.models.Namespace(**kwargs)`

Represents a BEL Namespace.

A simple constructor that allows initialization from kwargs.

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

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

uploaded

The date of upload

keyword

Keyword that is used in a BEL file to identify a specific namespace

pattern

Contains regex pattern for value identification.

miriam_id

MIRIAM resource identifier matching the regular expression `^MIR:001\d{5}$`

version

Version of the namespace

url

BELNS Resource location as URL

name

Name of the given namespace

domain

Domain for which this namespace is valid

species

Taxonomy identifiers for which this namespace is valid

description

Optional short description of the namespace

created

DateTime of the creation of the namespace definition file

query_url

URL that can be used to query the namespace (externally from PyBEL)

author

The author of the namespace

license

License information

contact

Contact information

get_term_to_encodings()

Return the term (db, id, name) to encodings from this namespace.

Return type `Mapping[Tuple[Optional[str], str], str]`

to_json(include_id=False)

Return the most useful entries as a dictionary.

Parameters `include_id (bool)` – If true, includes the model identifier

Return type `Mapping[str, str]`

class `pybel.manager.models.NamespaceEntry(**kwargs)`

Represents a name within a BEL namespace.

A simple constructor that allows initialization from kwargs.

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

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

name

Name that is defined in the corresponding namespace definition file

identifier

The database accession number

encoding

The biological entity types for which this name is valid

to_json(include_id=False)

Describe the namespaceEntry as dictionary of Namespace-Keyword and Name.

Parameters `include_id (bool)` – If true, includes the model identifier

Return type `Mapping[str, str]`

classmethod `name_contains(name_query)`

Make a filter if the name contains a certain substring.

class `pybel.manager.models.Network(**kwargs)`

Represents a collection of edges, specified by a BEL Script.

A simple constructor that allows initialization from kwargs.

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

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

name

Name of the given Network (from the BEL file)

version

Release version of the given Network (from the BEL file)

authors

Authors of the underlying BEL file

contact

Contact email from the underlying BEL file

description

Descriptive text from the underlying BEL file

copyright

Copyright information

disclaimer

Disclaimer information

licenses

License information

blob

A pickled version of this network

to_json(*include_id=False*)

Return this network as JSON.

Parameters **include_id** (*bool*) – If true, includes the model identifier

Return type *Mapping*[*str*, *Any*]

classmethod name_contains(*name_query*)

Build a filter for networks whose names contain the query.

classmethod description_contains(*description_query*)

Build a filter for networks whose descriptions contain the query.

classmethod id_in(*network_ids*)

Build a filter for networks whose identifiers appear in the given sequence.

as_bel()

Get this network and loads it into a BELGraph.

Return type *BELGraph*

store_bel(*graph*)

Insert a BEL graph.

class `pybel.manager.models.Node`(***kwargs*)

Represents a BEL Term.

A simple constructor that allows initialization from kwargs.

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

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

type

The type of the represented biological entity e.g. Protein or Gene

bel

Canonical BEL term that represents the given node

data

PyBEL BaseEntity as JSON

classmethod bel_contains(*bel_query*)

Build a filter for nodes whose BEL contain the query.

as_bel()

Serialize this node as a PyBEL DSL object.

Return type *pybel.dsl.BaseEntity*

to_json()

Serialize this node as a JSON object using as_bel().

class *pybel.manager.models*.**Author**(***kwargs*)

Contains all author names.

A simple constructor that allows initialization from kwargs.

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

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

classmethod name_contains(*name_query*)

Build a filter for authors whose names contain the given query.

classmethod has_name_in(*names*)

Build a filter if the author has any of the given names.

class *pybel.manager.models*.**Citation**(***kwargs*)

The information about the citations that are used to prove a specific relation are stored in this table.

A simple constructor that allows initialization from kwargs.

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

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

db

Type of the stored publication e.g. PubMed

db_id

Reference identifier of the publication e.g. PubMed_ID

article_type

Type of the publication

title

Title of the publication

journal

Journal name

volume

Volume of the journal

issue

Issue within the volume

pages

Pages of the publication

date

Publication date

first_id

First author

last_id

Last author

property is_pubmed: bool

Return if this is a PubMed citation.

Return type `bool`

property is_enriched: bool

Return if this citation has been enriched for name, title, and other metadata.

Return type `bool`

to_json(include_id=False)

Create a citation dictionary that is used to recreate the edge data dictionary of a BELGraph.

Parameters `include_id (bool)` – If true, includes the model identifier

Return type `Mapping[str, Any]`

Returns Citation dictionary for the recreation of a BELGraph.

class pybel.manager.models.Evidence(kwargs)**

This table contains the evidence text that proves a specific relationship and refers the source that is cited.

A simple constructor that allows initialization from kwargs.

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

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

text

Supporting text from a given publication

to_json(include_id=False)

Create a dictionary that is used to recreate the edge data dictionary for a BELGraph.

Parameters `include_id (bool)` – If true, includes the model identifier

Returns Dictionary containing citation and evidence for a BELGraph edge.

Return type `dict`

class pybel.manager.models.Edge(kwargs)**

Relationships between BEL nodes and their properties, annotations, and provenance.

A simple constructor that allows initialization from kwargs.

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

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

bel

Valid BEL statement that represents the given edge

source_modifier

Modifiers for the source of the edge

target_modifier

Modifiers for the target of the edge

md5

The hash of the source, target, and associated metadata

data

The stringified JSON representing this edge

get_annotations_json()

Format the annotations properly.

Return type Optional[dict[str,dict[str,bool]]]

to_json(include_id=False)

Create a dictionary of one BEL Edge that can be used to create an edge in a BELGraph.

Parameters **include_id** (*bool*) – Include the database identifier?

Return type Mapping[str, Any]

Returns Dictionary that contains information about an edge of a BELGraph. Including participants and edge data information.

insert_into_graph(graph)

Insert this edge into a BEL graph.

Return type str

An extensive set of examples can be found on the [PyBEL Notebooks](#) repository on GitHub. These notebooks contain basic usage and also make numerous references to the analytical package [PyBEL Tools](#)

22.1 Configuration

The default connection string can be set as an environment variable in your `~/.bashrc`. If you're using MySQL or MariaDB, it could look like this:

```
$ export PYBEL_CONNECTION="mysql+pymysql://user:password@server_name/database_name?  
↪charset=utf8"
```

22.1.1 Prepare a Cytoscape Network

Load, compile, and export to Cytoscape format:

```
$ pybel convert --path ~/Desktop/example.bel --graphml ~/Desktop/example.graphml
```

In Cytoscape, open with **Import > Network > From File**.

COMMAND LINE INTERFACE

Note: The command line wrapper might not work on Windows. Use `python3 -m pybel` if it has issues.

PyBEL automatically installs the command `pybel`. This command can be used to easily compile BEL documents and convert to other formats. See `pybel --help` for usage details. This command makes logs of all conversions and warnings to the directory `~/pybel/`.

23.1 pybel

PyBEL CLI on `/home/docs/checkouts/readthedocs.org/user_builds/pybel/envs/v0.15.4/bin/python`

```
pybel [OPTIONS] COMMAND [ARGS]...
```

Options

--version

Show the version and exit.

-c, --connection <connection>

Database connection string.

Default `sqlite:///home/docs/.data/pybel/pybel_0.14.0_cache.db`

23.1.1 compile

Compile a BEL script to a graph.

```
pybel compile [OPTIONS] PATH
```

Options

--allow-naked-names
Enable lenient parsing for naked names

--disallow-nested
Disable lenient parsing for nested statements

--disallow-unqualified-translocations
Disallow unqualified translocations

--no-identifier-validation
Turn off identifier validation

--no-citation-clearing
Turn off citation clearing

-r, --required-annotations <required_annotations>
Specify multiple required annotations

--upgrade-urls

--skip-tqdm

-v, --verbose

Arguments

PATH
Required argument

23.1.2 insert

Insert a graph to the database.

```
pybel insert [OPTIONS] path
```

Arguments

path
Required argument

23.1.3 machine

Get content from the INDRA machine and upload to BEL Commons.

```
pybel machine [OPTIONS] [AGENTS]...
```

Options

--local

Upload to local database.

--host <host>

URL of BEL Commons.

Arguments

AGENTS

Optional argument(s)

23.1.4 manage

Manage the database.

```
pybel manage [OPTIONS] COMMAND [ARGS]...
```

drop

Drop the database.

```
pybel manage drop [OPTIONS]
```

Options

--yes

Confirm the action without prompting.

edges

Manage edges.

```
pybel manage edges [OPTIONS] COMMAND [ARGS]...
```

ls

List edges.

```
pybel manage edges ls [OPTIONS]
```

Options

--offset <offset>

--limit <limit>

examples

Load examples to the database.

```
pybel manage examples [OPTIONS]
```

Options

-v, --debug

namespaces

Manage namespaces.

```
pybel manage namespaces [OPTIONS] COMMAND [ARGS]...
```

drop

Drop a namespace by URL.

```
pybel manage namespaces drop [OPTIONS] URL
```

Arguments

URL

Required argument

insert

Add a namespace by URL.

```
pybel manage namespaces insert [OPTIONS] URL
```

Arguments

URL

Required argument

ls

List cached namespaces.

```
pybel manage namespaces ls [OPTIONS]
```

Options

-u, --url <url>

Specific resource URL to list

-i, --namespace-id <namespace_id>

Specific resource URL to list

networks

Manage networks.

```
pybel manage networks [OPTIONS] COMMAND [ARGS]...
```

drop

Drop a network by its identifier or drop all networks.

```
pybel manage networks drop [OPTIONS]
```

Options

-n, --network-id <network_id>

Identifier of network to drop

-y, --yes

Drop all networks without confirmation if no identifier is given

ls

List network names, versions, and optionally, descriptions.

```
pybel manage networks ls [OPTIONS]
```

nodes

Manage nodes.

```
pybel manage nodes [OPTIONS] COMMAND [ARGS]...
```

prune

Prune nodes not belonging to any edges.

```
pybel manage nodes prune [OPTIONS]
```

summarize

Summarize the contents of the database.

```
pybel manage summarize [OPTIONS]
```

23.1.5 neo

Upload to neo4j.

```
pybel neo [OPTIONS] path
```

Options

--connection <connection>
Connection string for neo4j upload.

--password <password>

Arguments

path
Required argument

23.1.6 parse

Parse a single BEL statement and print JSON output.

```
pybel parse [OPTIONS] TEXT
```

Options

--pprint

Arguments

TEXT

Required argument

23.1.7 serialize

Serialize a graph to various formats.

```
pybel serialize [OPTIONS] path
```

Options

--tsv <tsv>

Path to output a TSV file.

--edgelist <edgelist>

Path to output a edgelist file.

--sif <sif>

Path to output an SIF file.

--gsea <gsea>

Path to output a GRP file for gene set enrichment analysis.

--graphml <graphml>

Path to output a GraphML file. Use .graphml for Cytoscape.

--nodelink <nodelink>

Path to output a node-link JSON file.

--bel <bel>

Output canonical BEL.

Arguments

path

Required argument

23.1.8 summarize

Summarize a graph.

```
pybel summarize [OPTIONS] path
```

Arguments

path

Required argument

23.1.9 upload

Upload a graph to BEL Commons.

```
pybel upload [OPTIONS] path
```

Options

--host <host>

URL of BEL Commons.

--user <user>

User for BEL Commons

Default <function _get_user at 0x7f94007b6290>

--password <password>

Password for BEL Commons

Default <function _get_password at 0x7f94007b6320>

Arguments

path

Required argument

23.1.10 warnings

List warnings from a graph.

```
pybel warnings [OPTIONS] path
```

Arguments

path

Required argument

CONSTANTS

Constants for PyBEL.

This module maintains the strings used throughout the PyBEL codebase to promote consistency.

`pybel.constants.get_cache_connection()`

Get the preferred RFC-1738 database connection string.

1. Check the environment variable `PYBEL_CONNECTION`
2. Check the `PYBEL_CONNECTION` key in the config file `~/.config/pybel/config.json`. Optionally, this config file might be in a different place if the environment variable `PYBEL_CONFIG_DIRECTORY` has been set.
3. Return a default connection string using a SQLite database in the `~/.pybel`. Optionally, this directory might be in a different place if the environment variable `PYBEL_RESOURCE_DIRECTORY` has been set.

Return type `str`

`pybel.constants.NAMESPACE_DOMAIN_TYPES = {'BiologicalProcess', 'Chemical', 'Gene and Gene Products', 'Other'}`

The valid namespace types .. seealso:: <https://wiki.openbel.org/display/BELNA/Custom+Namespaces>

`pybel.constants.CITATION_DATE = 'date'`

Represents the key for the citation date in a citation dictionary

`pybel.constants.CITATION_AUTHORS = 'authors'`

Represents the key for the citation authors in a citation dictionary

`pybel.constants.CITATION_JOURNAL = 'journal'`

Represents the key for the citation comment in a citation dictionary

`pybel.constants.CITATION_VOLUME = 'volume'`

Represents the key for the optional PyBEL citation volume entry in a citation dictionary

`pybel.constants.CITATION_ISSUE = 'issue'`

Represents the key for the optional PyBEL citation issue entry in a citation dictionary

`pybel.constants.CITATION_PAGES = 'pages'`

Represents the key for the optional PyBEL citation pages entry in a citation dictionary

`pybel.constants.CITATION_FIRST_AUTHOR = 'first'`

Represents the key for the optional PyBEL citation first author entry in a citation dictionary

`pybel.constants.CITATION_LAST_AUTHOR = 'last'`

Represents the key for the optional PyBEL citation last author entry in a citation dictionary

`pybel.constants.CITATION_ARTICLE_TYPE = 'article_type'`

Represents the type of article (Journal Article, Review, etc.)

`pybel.constants.FUNCTION = 'function'`

The node data key specifying the node's function (e.g. *GENE*, *MIRNA*, *BIOPROCESS*, etc.)

`pybel.constants.CONCEPT = 'concept'`

The key specifying a concept

`pybel.constants.NAMESPACE = 'namespace'`

The key specifying an identifier dictionary's namespace. Used for nodes, activities, and transformations.

`pybel.constants.NAME = 'name'`

The key specifying an identifier dictionary's name. Used for nodes, activities, and transformations.

`pybel.constants.IDENTIFIER = 'identifier'`

The key specifying an identifier dictionary

`pybel.constants.LABEL = 'label'`

The key specifying an optional label for the node

`pybel.constants.DESRIPTION = 'description'`

The key specifying an optional description for the node

`pybel.constants.XREFS = 'xref'`

The key specifying xrefs

`pybel.constants.MEMBERS = 'members'`

They key representing the nodes that are a member of a composite or complex

`pybel.constants.REACTANTS = 'reactants'`

The key representing the nodes appearing in the reactant side of a biochemical reaction

`pybel.constants.PRODUCTS = 'products'`

The key representing the nodes appearing in the product side of a biochemical reaction

`pybel.constants.PARTNER_3P = 'partner_3p'`

The key specifying the identifier dictionary of the fusion's 3-Prime partner

`pybel.constants.PARTNER_5P = 'partner_5p'`

The key specifying the identifier dictionary of the fusion's 5-Prime partner

`pybel.constants.RANGE_3P = 'range_3p'`

The key specifying the range dictionary of the fusion's 3-Prime partner

`pybel.constants.RANGE_5P = 'range_5p'`

The key specifying the range dictionary of the fusion's 5-Prime partner

`pybel.constants.VARIANTS = 'variants'`

The key specifying the node has a list of associated variants

`pybel.constants.KIND = 'kind'`

The key representing what kind of variation is being represented

`pybel.constants.HGVS = 'hgvs'`

The value for *KIND* for an HGVS variant

`pybel.constants.PMOD = 'pmod'`

The value for *KIND* for a protein modification

`pybel.constants.GMOD = 'gmod'`

The value for *KIND* for a gene modification

`pybel.constants.FRAGMENT = 'frag'`

The value for *KIND* for a fragment

```
pybel.constants.PYBEL_VARIANT_KINDS = {'frag', 'gmod', 'hgvs', 'pmod'}
```

The allowed values for *KIND*

```
pybel.constants.PYBEL_NODE_DATA_KEYS = {'function', 'fusion', 'identifier', 'members',
'name', 'namespace', 'products', 'reactants', 'variants'}
```

The group of all BEL-provided keys for node data dictionaries, used for hashing.

```
pybel.constants.DIRTY = 'dirty'
```

Used as a namespace when none is given when lenient parsing mode is turned on. Not recommended!

```
pybel.constants.ABUNDANCE = 'Abundance'
```

Represents the BEL abundance, abundance()

```
pybel.constants.GENE = 'Gene'
```

Represents the BEL abundance, geneAbundance() .. seealso:: http://openbel.org/language/version_2.0/bel_specification_version_2.0.html#Xabundancea

```
pybel.constants.RNA = 'RNA'
```

Represents the BEL abundance, rnaAbundance()

```
pybel.constants.MIRNA = 'miRNA'
```

Represents the BEL abundance, microRNAAbundance()

```
pybel.constants.PROTEIN = 'Protein'
```

Represents the BEL abundance, proteinAbundance()

```
pybel.constants.BIOPROCESS = 'BiologicalProcess'
```

Represents the BEL function, biologicalProcess()

```
pybel.constants.PATHOLOGY = 'Pathology'
```

Represents the BEL function, pathology()

```
pybel.constants.POPULATION = 'Population'
```

Represents the BEL function, populationAbundance()

```
pybel.constants.COMPOSITE = 'Composite'
```

Represents the BEL abundance, compositeAbundance()

```
pybel.constants.COMPLEX = 'Complex'
```

Represents the BEL abundance, complexAbundance()

```
pybel.constants.REACTION = 'Reaction'
```

Represents the BEL transformation, reaction()

```
pybel.constants.PYBEL_NODE_FUNCTIONS = {'Abundance', 'BiologicalProcess', 'Complex',
'Composite', 'Gene', 'Pathology', 'Population', 'Protein', 'RNA', 'Reaction', 'miRNA'}
```

A set of all of the valid PyBEL node functions

```
pybel.constants.rev_abundance_labels = {'Abundance': 'a', 'BiologicalProcess': 'bp',
'Complex': 'complex', 'Composite': 'composite', 'Gene': 'g', 'Pathology': 'path',
'Population': 'pop', 'Protein': 'p', 'RNA': 'r', 'miRNA': 'm'}
```

The mapping from PyBEL node functions to BEL strings

```
pybel.constants.RELATION = 'relation'
```

The key for an internal edge data dictionary for the relation string

```
pybel.constants.CITATION = 'citation'
```

The key for an internal edge data dictionary for the citation dictionary

```
pybel.constants.EVIDENCE = 'evidence'
```

The key for an internal edge data dictionary for the evidence string

```
pybel.constants.ANNOTATIONS = 'annotations'
```

The key for an internal edge data dictionary for the annotations dictionary

`pybel.constants.SOURCE_MODIFIER = 'source_modifier'`

The key for an internal edge data dictionary for the source modifier dictionary

`pybel.constants.TARGET_MODIFIER = 'target_modifier'`

The key for an internal edge data dictionary for the target modifier dictionary

`pybel.constants.LINE = 'line'`

The key or an internal edge data dictionary for the line number

`pybel.constants.HASH = 'hash'`

The key representing the hash of the other

`pybel.constants.PYBEL_EDGE_DATA_KEYS = {'annotations', 'citation', 'evidence', 'relation', 'source_modifier', 'target_modifier'}`

The group of all BEL-provided keys for edge data dictionaries, used for hashing.

`pybel.constants.PYBEL_EDGE_METADATA_KEYS = {'hash', 'line'}`

The group of all PyBEL-specific keys for edge data dictionaries, not used for hashing.

`pybel.constants.PYBEL_EDGE_ALL_KEYS = {'annotations', 'citation', 'evidence', 'hash', 'line', 'relation', 'source_modifier', 'target_modifier'}`

The group of all PyBEL annotated keys for edge data dictionaries

`pybel.constants.HAS_REACTANT = 'hasReactant'`

A BEL relationship

`pybel.constants.HAS_PRODUCT = 'hasProduct'`

A BEL relationship

`pybel.constants.HAS_VARIANT = 'hasVariant'`

A BEL relationship

`pybel.constants.TRANSCRIBED_TO = 'transcribedTo'`

A BEL relationship *GENE* to *RNA* is called transcription

`pybel.constants.TRANSLATED_TO = 'translatedTo'`

A BEL relationship *RNA* to *PROTEIN* is called translation

`pybel.constants.INCREASES = 'increases'`

A BEL relationship

`pybel.constants.DIRECTLY_INCREASES = 'directlyIncreases'`

A BEL relationship

`pybel.constants.DECREASES = 'decreases'`

A BEL relationship

`pybel.constants.DIRECTLY_DECREASES = 'directlyDecreases'`

A BEL relationship

`pybel.constants.CAUSES_NO_CHANGE = 'causesNoChange'`

A BEL relationship

`pybel.constants.REGULATES = 'regulates'`

A BEL relationship

`pybel.constants.DIRECTLY_REGULATES = 'directlyRegulates'`

A BEL relationship

`pybel.constants.BINDS = 'binds'`

A BEL relationship

`pybel.constants.CORRELATION = 'correlation'`

A BEL relationship

```

pybel.constants.NO_CORRELATION = 'noCorrelation'
    A BEL relationship

pybel.constants.NEGATIVE_CORRELATION = 'negativeCorrelation'
    A BEL relationship

pybel.constants.POSITIVE_CORRELATION = 'positiveCorrelation'
    A BEL relationship

pybel.constants.ASSOCIATION = 'association'
    A BEL relationship

pybel.constants.ORTHOLOGOUS = 'orthologous'
    A BEL relationship

pybel.constants.ANALOGOUS_TO = 'analogousTo'
    A BEL relationship

pybel.constants.IS_A = 'isA'
    A BEL relationship

pybel.constants.RATE_LIMITING_STEP_OF = 'rateLimitingStepOf'
    A BEL relationship

pybel.constants.SUBPROCESS_OF = 'subProcessOf'
    A BEL relationship

pybel.constants.BIOMARKER_FOR = 'biomarkerFor'
    A BEL relationship

pybel.constants.PROGNOSTIC_BIOMARKER_FOR = 'prognosticBiomarkerFor'
    A BEL relationship

pybel.constants.EQUIVALENT_TO = 'equivalentTo'
    A BEL relationship, added by PyBEL

pybel.constants.PART_OF = 'partOf'
    A BEL relationship, added by PyBEL

pybel.constants.CAUSAL_INCREASE_RELATIONS = {'directlyIncreases', 'increases'}
    A set of all causal relationships that have an increasing effect

pybel.constants.CAUSAL_DECREASE_RELATIONS = {'decreases', 'directlyDecreases'}
    A set of all causal relationships that have a decreasing effect

pybel.constants.CAUSAL_APOLAR_RELATIONS = {'directlyRegulates', 'regulates'}
    A set of all causal relationships that have an indeterminate polarity

pybel.constants.DIRECT_CAUSAL_RELATIONS = {'directlyDecreases', 'directlyIncreases',
'directlyRegulates'}
    A set of direct causal relations

pybel.constants.INDIRECT_CAUSAL_RELATIONS = {'decreases', 'increases', 'regulates'}
    A set of indirect causal relations

pybel.constants.CAUSAL_POLAR_RELATIONS = {'decreases', 'directlyDecreases',
'directlyIncreases', 'increases'}
    A set of causal relationships that are polar

pybel.constants.CAUSAL_RELATIONS = {'decreases', 'directlyDecreases',
'directlyIncreases', 'directlyRegulates', 'increases', 'regulates'}
    A set of all causal relationships

```

```
pybel.constants.CORRELATIVE_RELATIONS = {'correlation', 'negativeCorrelation',  
'noCorrelation', 'positiveCorrelation'}
```

A set of all correlative relationships

```
pybel.constants.POLAR_RELATIONS = {'decreases', 'directlyDecreases', 'directlyIncreases',  
'increases', 'negativeCorrelation', 'positiveCorrelation'}
```

A set of polar relations

```
pybel.constants.TWO_WAY_RELATIONS = {'analogousTo', 'association', 'binds',  
'correlation', 'equivalentTo', 'negativeCorrelation', 'noCorrelation', 'orthologous',  
'positiveCorrelation'}
```

A set of all relationships that are inherently directionless, and are therefore added to the graph twice

```
pybel.constants.UNQUALIFIED_EDGES = {'equivalentTo', 'hasProduct', 'hasReactant',  
'hasVariant', 'isA', 'orthologous', 'partOf', 'transcribedTo', 'translatedTo'}
```

A list of relationship types that don't require annotations or evidence

```
pybel.constants.GRAPH_METADATA = 'document_metadata'
```

The key for the document metadata dictionary. Can be accessed by `graph.graph[GRAPH_METADATA]`, or by using the property built in to the [pybel.BELGraph](#), `pybel.BELGraph.document()`

```
pybel.constants.METADATA_NAME = 'name'
```

The key for the document name. Can be accessed by `graph.document[METADATA_NAME]` or by using the property built into the [pybel.BELGraph](#) class, [pybel.BELGraph.name\(\)](#)

```
pybel.constants.METADATA_VERSION = 'version'
```

The key for the document version. Can be accessed by `graph.document[METADATA_VERSION]`

```
pybel.constants.METADATA_DESCRIPTION = 'description'
```

The key for the document description. Can be accessed by `graph.document[METADATA_DESCRIPTION]`

```
pybel.constants.METADATA_AUTHORS = 'authors'
```

The key for the document authors. Can be accessed by `graph.document[METADATA_NAME]`

```
pybel.constants.METADATA_CONTACT = 'contact'
```

The key for the document contact email. Can be accessed by `graph.document[METADATA_CONTACT]`

```
pybel.constants.METADATA_LICENSES = 'licenses'
```

The key for the document licenses. Can be accessed by `graph.document[METADATA_LICENSES]`

```
pybel.constants.METADATA_COPYRIGHT = 'copyright'
```

The key for the document copyright information. Can be accessed by `graph.document[METADATA_COPYRIGHT]`

```
pybel.constants.METADATA_DISCLAIMER = 'disclaimer'
```

The key for the document disclaimer. Can be accessed by `graph.document[METADATA_DISCLAIMER]`

```
pybel.constants.METADATA_PROJECT = 'project'
```

The key for the document project. Can be accessed by `graph.document[METADATA_PROJECT]`

```
pybel.constants.DOCUMENT_KEYS = {'Authors': 'authors', 'ContactInfo': 'contact',  
'Copyright': 'copyright', 'Description': 'description', 'Disclaimer': 'disclaimer',  
'Licenses': 'licenses', 'Name': 'name', 'Project': 'project', 'Version': 'version'}
```

Provides a mapping from BEL language keywords to internal PyBEL strings

```
pybel.constants.METADATA_INSERT_KEYS = {'authors', 'contact', 'copyright', 'description',  
'disclaimer', 'licenses', 'name', 'version'}
```

The keys to use when inserting a graph to the cache

```
pybel.constants.INVERSE_DOCUMENT_KEYS = {'authors': 'Authors', 'contact':
'ContactInfo', 'copyright': 'Copyright', 'description': 'Description', 'disclaimer':
'Disclaimer', 'licenses': 'Licenses', 'name': 'Name', 'project': 'Project', 'version':
'Version'}
```

Provides a mapping from internal PyBEL strings to BEL language keywords. Is the inverse of [DOCUMENT_KEYS](#)

```
pybel.constants.REQUIRED_METADATA = {'authors', 'contact', 'description', 'name',
'version'}
```

A set representing the required metadata during BEL document parsing

```
pybel.constants.FRAGMENT_START = 'start'
```

The key for the starting position of a fragment range

```
pybel.constants.FRAGMENT_STOP = 'stop'
```

The key for the stopping position of a fragment range

```
pybel.constants.FRAGMENT_MISSING = 'missing'
```

The key signifying that there is neither a start nor stop position defined

```
pybel.constants.FRAGMENT_DESCRIPTION = 'description'
```

The key for any additional descriptive data about a fragment

```
pybel.constants.GMOD_ORDER = ['kind', 'identifier']
```

The order for serializing gene modification data

```
pybel.constants.GSUB_REFERENCE = 'reference'
```

The key for the reference nucleotide in a gene substitution. Only used during parsing since this is converted to HGVS.

```
pybel.constants.GSUB_POSITION = 'position'
```

The key for the position of a gene substitution. Only used during parsing since this is converted to HGVS

```
pybel.constants.GSUB_VARIANT = 'variant'
```

The key for the effect of a gene substitution. Only used during parsing since this is converted to HGVS

```
pybel.constants.PMOD_CODE = 'code'
```

The key for the protein modification code.

```
pybel.constants.PMOD_POSITION = 'pos'
```

The key for the protein modification position.

```
pybel.constants.PMOD_ORDER = ['kind', 'identifier', 'code', 'pos']
```

The order for serializing information about a protein modification

```
pybel.constants.PSUB_REFERENCE = 'reference'
```

The key for the reference amino acid in a protein substitution. Only used during parsing since this is converted to HGVS

```
pybel.constants.PSUB_POSITION = 'position'
```

The key for the position of a protein substitution. Only used during parsing since this is converted to HGVS.

```
pybel.constants.PSUB_VARIANT = 'variant'
```

The key for the variant of a protein substitution. Only used during parsing since this is converted to HGVS.

```
pybel.constants.TRUNCATION_POSITION = 'position'
```

The key for the position at which a protein is truncated

```
pybel.constants.belns_encodings = {'A': {'Abundance', 'Complex', 'Gene', 'Protein',
'RNA', 'miRNA'}, 'B': {'BiologicalProcess', 'Pathology'}, 'C': {'Complex'}, 'G':
{'Gene'}, 'M': {'miRNA'}, 'O': {'Pathology'}, 'P': {'Protein'}, 'R': {'RNA', 'miRNA'}}
```

The mapping from BEL namespace codes to PyBEL internal abundance constants ..seealso:: <https://wiki.openbel.org/display/BELNA/Assignment+of+Encoding+%28Allowed+Functions%29+for+BEL+Namespaces>

Language constants for BEL.

This module contains mappings between PyBEL's internal constants and BEL language keywords.

class `pybel.language.Entity`(*, *namespace*, *name=None*, *identifier=None*)

Represents a named entity with a namespace and name/identifier.

Create a dictionary representing a reference to an entity.

Parameters

- **namespace** (`str`) – The namespace to which the entity belongs
- **name** (`Optional[str]`) – The name of the entity
- **identifier** (`Optional[str]`) – The identifier of the entity in the namespace

property namespace: `str`

The entity's namespace.

Return type `str`

property name: `str`

The entity's name or label.

Return type `str`

property identifier: `str`

The entity's identifier.

Return type `str`

property curie: `str`

Return this entity as a CURIE.

Return type `str`

property obo: `str`

Return this entity as an OBO-style CURIE.

Return type `str`

```
pybel.language.activity_labels = {'cat': 'cat', 'catalyticActivity': 'cat', 'chap':  
'chap', 'chaperoneActivity': 'chap', 'gap': 'gap', 'gef': 'gef', 'gtp': 'gtp',  
'gtpBoundActivity': 'gtp', 'gtpaseActivatingProteinActivity': 'gap',  
'guanineNucleotideExchangeFactorActivity': 'gef', 'kin': 'kin', 'kinaseActivity':  
'kin', 'molecularActivity': 'molecularActivity', 'pep': 'pep', 'peptidaseActivity':  
'pep', 'phos': 'phos', 'phosphataseActivity': 'phos', 'ribo': 'ribo',  
'ribosylationActivity': 'ribo', 'tport': 'tport', 'transcriptionalActivity':  
'tscript', 'transportActivity': 'tport', 'tscript': 'tscript'}
```

A dictionary of activity labels used in the `ma()` function in `activity(p(X), ma(Y))`


```
pybel.language.activity_mapping = {'act': {'identifier': '0003674', 'name': 'molecular
function', 'namespace': 'go'}, 'cat': {'identifier': '0003824', 'name': 'catalytic
activity', 'namespace': 'go'}, 'chap': {'identifier': '0044183', 'name': 'protein
binding involved in protein folding', 'namespace': 'go'}, 'gap': {'identifier':
'0032794', 'name': 'GTPase activating protein binding', 'namespace': 'go'}, 'gef':
{'identifier': '0005085', 'name': 'guanyl-nucleotide exchange factor activity',
'namespace': 'go'}, 'gtp': {'identifier': '0005525', 'name': 'GTP binding',
'namespace': 'go'}, 'kin': {'identifier': '0016301', 'name': 'kinase activity',
'namespace': 'go'}, 'molecularActivity': {'identifier': '0003674', 'name':
'molecular_function', 'namespace': 'go'}, 'pep': {'identifier': '0008233', 'name':
'peptidase activity', 'namespace': 'go'}, 'phos': {'identifier': '0016791', 'name':
'phosphatase activity', 'namespace': 'go'}, 'ribo': {'identifier': '0003956', 'name':
'NAD(P)+-protein-arginine ADP-ribosyltransferase activity', 'namespace': 'go'}, 'tport':
{'identifier': '0005215', 'name': 'transporter activity', 'namespace': 'go'},
'tscript': {'identifier': '0001071', 'name': 'nucleic acid binding transcription
factor activity', 'namespace': 'go'}}
```

Maps the default BEL molecular activities to Gene Ontology Molecular Functions

```
pybel.language.compartment_mapping = {'cell surface': {'identifier': '0009986', 'name':
'cell surface', 'namespace': 'go'}, 'cytoplasm': {'identifier': '0005737', 'name':
'cytoplasm', 'namespace': 'go'}, 'extracellular space': {'identifier': '0005615',
'name': 'extracellular space', 'namespace': 'go'}, 'intracellular': {'identifier':
'0005622', 'name': 'intracellular', 'namespace': 'go'}, 'nucleus': {'identifier':
'0005634', 'name': 'nucleus', 'namespace': 'go'}}
```

Maps the default BEL cellular components to Gene Ontology Cellular Components

```
pybel.language.abundance_labels = {'a': 'Abundance', 'abundance': 'Abundance',
'biologicalProcess': 'BiologicalProcess', 'bp': 'BiologicalProcess', 'complex':
'Complex', 'complexAbundance': 'Complex', 'composite': 'Composite',
'compositeAbundance': 'Composite', 'g': 'Gene', 'geneAbundance': 'Gene', 'm':
'miRNA', 'microRNAAbundance': 'miRNA', 'p': 'Protein', 'path': 'Pathology',
'pathology': 'Pathology', 'proteinAbundance': 'Protein', 'r': 'RNA', 'rnaAbundance':
'RNA'}
```

Provides a mapping from BEL terms to PyBEL internal constants

```
pybel.language.abundance_sbo_mapping = {'BiologicalProcess': {'identifier': '0000375',
'name': 'process', 'namespace': 'sbo'}, 'Complex': {'identifier': '0000297', 'name':
'protein complex', 'namespace': 'sbo'}, 'Gene': {'identifier': '0000243', 'name':
'gene', 'namespace': 'sbo'}, 'Pathology': {'identifier': '0000358', 'name':
'phenotype', 'namespace': 'sbo'}, 'RNA': {'identifier': '0000278', 'name': 'messenger
RNA', 'namespace': 'sbo'}, 'miRNA': {'identifier': '0000316', 'name': 'microRNA',
'namespace': 'sbo'}}
```

Maps the BEL abundance types to the Systems Biology Ontology

```
pybel.language.pmod_namespace = {'ADP-ribosylation': 'ADPRib', 'ADPRib': 'ADPRib',  
'Ac': 'Ac', 'Farn': 'Farn', 'Gerger': 'Gerger', 'Glyco': 'Glyco', 'Hy': 'Hy', 'ISG':  
'ISG', 'ISG15-protein conjugation': 'ISG', 'ISGylation': 'ISG', 'Lysine 48-linked  
polyubiquitination': 'UbK48', 'Lysine 63-linked polyubiquitination': 'UbK63', 'Me':  
'Me', 'Me1': 'Me1', 'Me2': 'Me2', 'Me3': 'Me3', 'Myr': 'Myr', 'N-linked  
glycosylation': 'NGlyco', 'NGlyco': 'NGlyco', 'NO': 'NO', 'Nedd': 'Nedd',  
'Nitrosylation': 'NO', 'O-linked glycosylation': 'OGlyco', 'OGlyco': 'OGlyco', 'Ox':  
'Ox', 'Palm': 'Palm', 'Ph': 'Ph', 'SUMOylation': 'Sumo', 'Sulf': 'Sulf', 'Sumo':  
'Sumo', 'Ub': 'Ub', 'UbK48': 'UbK48', 'UbK63': 'UbK63', 'UbMono': 'UbMono', 'UbPoly':  
'UbPoly', 'acetylation': 'Ac', 'adenosine diphosphoribosyl': 'ADPRib',  
'di-methylation': 'Me2', 'dimethylation': 'Me2', 'farnesylation': 'Farn',  
'geranylgeranylation': 'Gerger', 'glycosylation': 'Glyco', 'hydroxylation': 'Hy',  
'methylation': 'Me', 'mono-methylation': 'Me1', 'monomethylation': 'Me1',  
'monoubiquitination': 'UbMono', 'myristoylation': 'Myr', 'neddylation': 'Nedd',  
'oxidation': 'Ox', 'palmitoylation': 'Palm', 'phosphorylation': 'Ph',  
'polyubiquitination': 'UbPoly', 'sulfation': 'Sulf', 'sulfonation': 'sulfonation',  
'sulfur addition': 'Sulf', 'sulphation': 'Sulf', 'sulphonation': 'sulfonation',  
'sulphur addition': 'Sulf', 'tri-methylation': 'Me3', 'trimethylation': 'Me3',  
'ubiquitination': 'Ub', 'ubiquitinylation': 'Ub', 'ubiquitylation': 'Ub'}
```

A dictionary of default protein modifications to their preferred value

```

pybel.language.pmod_mappings = {'ADPRib': {'synonyms': ['ADPRib', 'ADP-ribosylation',
'ADPRib', 'ADP-rybosylation', 'adenosine diphosphoribosyl'], 'xrefs': [{'namespace':
'go', 'name': 'protein ADP-ribosylation', 'identifier': '0006471'}, {'namespace':
'mod', 'name': 'adenosine diphosphoribosyl (ADP-ribosyl) modified residue',
'identifier': '00752'}, {'namespace': 'mop', 'name': 'adenosinediphosphoribosylation',
'identifier': '0000220'}]}, 'Ac': {'synonyms': ['Ac', 'acetylation'], 'xrefs':
[{'namespace': 'go', 'name': 'protein acetylation', 'identifier': '0006473'},
{'namespace': 'mod', 'name': 'acetylated residue', 'identifier': '00394'},
{'namespace': 'mop', 'name': 'acetylation', 'identifier': '0000030'}, {'namespace':
'sbo', 'name': 'acetylation', 'identifier': '0000215'}]}, 'Farn': {'synonyms':
['Farn', 'farnesylation'], 'xrefs': [{'namespace': 'go', 'name': 'protein
farnesylation', 'identifier': '0018343'}, {'namespace': 'mod', 'name': 'farnesylated
residue', 'identifier': '00437'}, {'namespace': 'mop', 'name': 'farnesylation',
'identifier': '0000429'}]}, 'Gerger': {'synonyms': ['Gerger', 'geranylgeranylation'],
'xrefs': [{'namespace': 'go', 'name': 'protein geranylgeranylation', 'identifier':
'0018344'}, {'namespace': 'mod', 'name': 'geranylgeranylated residue', 'identifier':
'00441'}, {'namespace': 'mop', 'name': 'geranylgeranylation', 'identifier':
'0000431'}]}, 'Glyco': {'synonyms': ['Glyco', 'glycosylation'], 'xrefs':
[{'namespace': 'go', 'name': 'protein glycosylation', 'identifier': '0006486'},
{'namespace': 'mod', 'name': 'glycosylated residue', 'identifier': '00693'},
{'namespace': 'mop', 'name': 'glycosylation', 'identifier': '0000162'}]}, 'Hy':
{'synonyms': ['Hyhydroxylation'], 'xrefs': [{'namespace': 'go', 'name': 'protein
hydroxylation', 'identifier': '0018126'}, {'namespace': 'mod', 'name': 'hydroxylated
residue', 'identifier': '00677'}, {'namespace': 'mop', 'name': 'hydroxylation',
'identifier': '0000673'}]}, 'ISG': {'activities': [{'namespace': 'go', 'name': 'ISG15
transferase activity', 'identifier': '0042296'}], 'synonyms': ['ISG', 'ISGylation',
'ISG15-protein conjugation'], 'xrefs': [{'namespace': 'go', 'name': 'ISG15-protein
conjugation', 'identifier': '0032020'}]}, 'Me': {'synonyms': ['Me', 'methylation'],
'xrefs': [{'namespace': 'go', 'name': 'protein methylation', 'identifier':
'0006479'}, {'namespace': 'mod', 'name': 'methylated residue', 'identifier':
'00427'}]}, 'Me1': {'is_a': ['Me'], 'synonyms': ['Me1', 'monomethylation',
'mono-methylation'], 'xrefs': [{'namespace': 'mod', 'name': 'monomethylated residue',
'identifier': '00599'}]}, 'Me2': {'is_a': ['Me'], 'synonyms': ['Me2',
'dimethylation', 'di-methylation'], 'xrefs': [{'namespace': 'mod', 'name':
'dimethylated residue', 'identifier': '00429'}]}, 'Me3': {'is_a': ['Me'], 'synonyms':
['Me3', 'trimethylation', 'tri-methylation'], 'xrefs': [{'namespace': 'mod', 'name':
'trimethylated residue', 'identifier': '00430'}]}, 'Myr': {'synonyms': ['Myr',
'myristoylation'], 'xrefs': [{'namespace': 'go', 'name': 'protein myristoylation',
'identifier': '0018377'}, {'namespace': 'mod', 'name': 'myristoylated residue',
'identifier': '00438'}]}, 'NGlyco': {'is_a': ['Glyco'], 'synonyms': ['NGlyco',
'N-linked glycosylation'], 'xrefs': [{'namespace': 'go', 'name': 'protein N-linked
glycosylation', 'identifier': '0006487'}, {'namespace': 'mod', 'name': 'N-glycosylated
residue', 'identifier': '000006'}, {'namespace': 'mop', 'name': 'N-glycosylation',
'identifier': '0002162'}]}, 'NO': {'synonyms': ['NO', 'Nitrosylation'], 'xrefs':
[{'namespace': 'go', 'name': 'protein nitrosylation', 'identifier': '0017014'}]},
'Nedd': {'synonyms': ['Nedd', 'neddylation', 'RUB1-protein conjugation'], 'xrefs':
[{'namespace': 'go', 'name': 'protein neddylation', 'identifier': '0045116'},
{'namespace': 'mod', 'name': 'neddylated lysine', 'identifier': '01150'}]}, 'OGlyco':
{'is_a': ['Glyco'], 'synonyms': ['OGlyco', 'O-linked glycosylation'], 'xrefs':
[{'namespace': 'go', 'name': 'protein O-linked glycosylation', 'identifier':
'0006493'}, {'namespace': 'mod', 'name': 'O-glycosylated residue', 'identifier':
'00396'}, {'namespace': 'mop', 'name': 'O-glycosylation', 'identifier': '0003162'}]},
'Ox': {'synonyms': ['Ox', 'oxidation'], 'xrefs': [{'namespace': 'go', 'name':
'protein oxidation', 'identifier': '0018158'}]}, 'Palm': {'synonyms': ['Palm',
'palmitoylation'], 'xrefs': [{'namespace': 'go', 'name': 'protein palmitoylation',
'identifier': '0018345'}, {'namespace': 'mod', 'name': 'palmitoylated residue',
'identifier': '00440'}]}, 'Ph': {'synonyms': ['Ph', 'phosphorylation'], 'xrefs': 149
[{'namespace': 'go', 'name': 'protein phosphorylation', 'identifier': '0006468'},
{'namespace': 'mod', 'identifier': '00696'}]}, 'Sulf': {'synonyms': ['Sulf',
'sulfation', 'sulphation', 'sulfur addition', 'sulphur addition', 'sulfonation',

```

Use Gene Ontology children of go_0006464: “cellular protein modification process”

```
pybel.language.pmod_legacy_labels = {'A': 'Ac', 'F': 'Farn', 'G': 'Glyco', 'H': 'Hy',  
                                     'M': 'Me', 'O': 'Ox', 'P': 'Ph', 'R': 'ADPRib', 'S': 'Sumo', 'U': 'Ub'}
```

A dictionary of legacy (BEL 1.0) default namespace protein modifications to their BEL 2.0 preferred value

```
pybel.language.gmod_namespace = {'ADPRib': 'ADPRib', 'M': 'Me', 'Me': 'Me',  
                                 'methylation': 'Me'}
```

A dictionary of default gene modifications. This is a PyBEL variant to the BEL specification.

```
pybel.language.gmod_mappings = {'ADPRib': {'synonyms': ['ADPRib'], 'xrefs':  
[{'namespace': 'go', 'name': 'DNA ADP-ribosylation', 'identifier': '0030592'}]}, 'Me':  
{'synonyms': ['Me', 'M', 'methylation'], 'xrefs': [{'namespace': 'go', 'name': 'DNA  
methylation', 'identifier': '0006306'}]}}
```

Use Gene Ontology children of [go:0006304](#) ! “DNA modification”

```
class pybel.language.CitationDict(namespace, identifier, *, name=None, **kwargs)
```

A dictionary describing a citation.

Create a dictionary representing a reference to an entity.

Parameters

- **namespace** ([str](#)) – The namespace to which the entity belongs
- **name** ([Optional\[str\]](#)) – The name of the entity
- **identifier** ([str](#)) – The identifier of the entity in the namespace

```
pybel.language.citation_dict(*, namespace=None, db=None, identifier=None, db_id=None, name=None,  
                             **kwargs)
```

Make a citation dictionary.

Return type [CitationDict](#)

PARSERS

This page is for users who want to squeeze the most bizarre possibilities out of PyBEL. Most users will not need this reference.

PyBEL makes extensive use of the PyParsing module. The code is organized to different modules to reflect the different faces of the BEL language. These parsers support BEL 2.0 and have some backwards compatibility for rewriting BEL v1.0 statements as BEL v2.0. The biologist and bioinformatician using this software will likely never need to read this page, but a developer seeking to extend the language will be interested to see the inner workings of these parsers.

See: https://github.com/OpenBEL/language/blob/master/version_2.0/MIGRATE_BEL1_BEL2.md

25.1 BEL Parser

```
class pybel.parser.parse_bel.BELParser(graph=None, namespace_to_term_to_encoding=None,
                                       namespace_to_pattern=None, annotation_to_term=None,
                                       annotation_to_pattern=None, annotation_to_local=None,
                                       allow_naked_names=False, disallow_nested=False,
                                       disallow_unqualified_translocations=False,
                                       citation_clearing=True, skip_validation=False,
                                       autostreamline=True, required_annotations=None)
```

Build a parser backed by a given dictionary of namespaces.

Build a BEL parser.

Parameters

- **graph** (*Optional*[*BELGraph*]) – The BEL graph to use to store the network
- **namespace_to_term_to_encoding** (*Optional*[*Mapping*[*str*, *Mapping*[*Tuple*[*Optional*[*str*], *str*], *str*]]]) – A dictionary of {namespace: {name: encoding}}. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **namespace_to_pattern** (*Optional*[*Mapping*[*str*, *Pattern*]]]) – A dictionary of {namespace: compiled regular expression}. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **annotation_to_term** (*Optional*[*Mapping*[*str*, *Set*[*str*]]]) – A dictionary of {annotation: set of values}. Delegated to `pybel.parser.ControlParser`
- **annotation_to_pattern** (*Optional*[*Mapping*[*str*, *Pattern*]]]) – A dictionary of {annotation: regular expression strings}. Delegated to `pybel.parser.ControlParser`
- **annotation_to_local** (*Optional*[*Mapping*[*str*, *Set*[*str*]]]) – A dictionary of {annotation: set of values}. Delegated to `pybel.parser.ControlParser`

- **allow_naked_names** (*bool*) – If true, turn off naked namespace failures. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **disallow_nested** (*bool*) – If true, turn on nested statement failures. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **disallow_unqualified_translocations** (*bool*) – If true, allow translocations without TO and FROM clauses.
- **citation_clearing** (*bool*) – Should SET Citation statements clear evidence and all annotations? Delegated to `pybel.parser.ControlParser`
- **autostreamline** (*bool*) – Should the parser be streamlined on instantiation?
- **required_annotations** (*Optional[List[str]]*) – Optional list of required annotations

pmod

2.2.1

location

2.2.4

gmod

PyBEL BEL Specification variant

fusion

2.6.1

general_abundance

2.1.1

gene

2.1.4

mirna

2.1.5

protein

2.1.6

rna

2.1.7

complex_singleton

2.1.2

composite_abundance

2.1.3

molecular_activity

2.4.1

biological_process

2.3.1

pathology

2.3.2

activity

2.3.3

translocation

2.5.1

degradation

2.5.2

reactants

2.5.3

rate_limit

3.1.5

subprocess_of

3.4.6

transcribed

3.3.2

translated

3.3.3

has_member

3.4.1

abundance_list

3.4.2

parse(s)

Parse the string.

Return type Mapping[str, Any]**get_annotations()**

Get the current annotations in this parser.

Return type Dict**clear()**

Clear the graph and all control parser data (current citation, annotations, and statement group).

handle_nested_relation(line, position, tokens)

Handle nested statements.

If self.disallow_nested is True, raises a NestedRelationWarning.

Raises NestedRelationWarning**check_function_semantics(line, position, tokens)**

Raise an exception if the function used on the tokens is wrong.

Raises InvalidFunctionSemantic**Return type** ParseResults**handle_term(_, __, tokens)**

Handle BEL terms (the subject and object of BEL relations).

Return type ParseResults**handle_has_members(_, __, tokens)**

Handle list relations like p(X) hasMembers list(p(Y), p(Z), ...).

Return type ParseResults**handle_has_components(_, __, tokens)**

Handle list relations like p(X) hasComponents list(p(Y), p(Z), ...).

Return type ParseResults

handle_unqualified_relation(_, __, *tokens*)

Handle unqualified relations.

Return type `ParseResults`

handle_inverse_unqualified_relation(_, __, *tokens*)

Handle unqualified relations that should go reverse.

Return type `ParseResults`

ensure_node(*tokens*)

Turn parsed tokens into canonical node name and makes sure its in the graph.

Return type `BaseEntity`

handle_translocation_illegal(*line*, *position*, *tokens*)

Handle a malformed translocation.

Return type `None`

`pybel.io.line_utils.parse_lines`(*graph*, *lines*, *manager=None*, *disallow_nested=False*,
citation_clearing=True, *use_tqdm=False*, *tqdm_kwargs=None*,
no_identifier_validation=False,
disallow_unqualified_translocations=False, *allow_redefinition=False*,
allow_definition_failures=False, *allow_naked_names=False*,
required_annotations=None, *upgrade_urls=False*)

Parse an iterable of lines into this graph.

Delegates to `parse_document()`, `parse_definitions()`, and `parse_statements()`.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **lines** (`Iterable[str]`) – An iterable over lines of BEL script
- **manager** (`Optional[Manager]`) – A PyBEL database manager
- **disallow_nested** (`bool`) – If true, turns on nested statement failures
- **citation_clearing** (`bool`) – Should SET Citation statements clear evidence and all annotations? Delegated to `pybel.parser.ControlParser`
- **use_tqdm** (`bool`) – Use `tqdm` to show a progress bar?
- **tqdm_kwargs** (`Optional[Mapping[str, Any]]`) – Keywords to pass to `tqdm`
- **disallow_unqualified_translocations** (`bool`) – If true, allow translocations without TO and FROM clauses.
- **required_annotations** (`Optional[List[str]]`) – Annotations that are required for all statements
- **upgrade_urls** (`bool`) – Automatically upgrade old namespace URLs. Defaults to false.

Warning: These options allow concessions for parsing BEL that is either **WRONG** or **UNSCIENTIFIC**. Use them at risk to reproducibility and validity of your results.

Parameters

- **no_identifier_validation** (`bool`) – If true, turns off namespace validation
- **allow_naked_names** (`bool`) – If true, turns off naked namespace failures

- **allow_redefinition** (*bool*) – If true, doesn't fail on second definition of same name or annotation
- **allow_definition_failures** (*bool*) – If true, allows parsing to continue if a terminology file download/parse fails

Return type *None*

25.2 Metadata Parser

```
class pybel.parser.parse_metadata.MetadataParser(manager, namespace_to_term_to_encoding=None,
                                                namespace_to_pattern=None,
                                                annotation_to_term=None,
                                                annotation_to_pattern=None,
                                                annotation_to_local=None,
                                                default_namespace=None,
                                                allow_redefinition=False, skip_validation=False,
                                                upgrade_urls=False)
```

A parser for the document and definitions section of a BEL document.

See also:

BEL 1.0 Specification for the [DEFINE](#) keyword

Build a metadata parser.

Parameters

- **manager** – A cache manager
- **namespace_to_term_to_encoding** (*Optional[Mapping[str, Mapping[Tuple[Optional[str], str], str]]]*) – An enumerated namespace mapping from {namespace keyword: {(identifier, name): encoding}}
- **namespace_to_pattern** (*Optional[Mapping[str, Pattern]]*) – A regular expression namespace mapping from {namespace keyword: regex string}
- **annotation_to_term** (*Optional[Mapping[str, Set[str]]]*) – Enumerated annotation mapping from {annotation keyword: set of valid values}
- **annotation_to_pattern** (*Optional[Mapping[str, Pattern]]*) – Regular expression annotation mapping from {annotation keyword: regex string}
- **default_namespace** (*Optional[Set[str]]*) – A set of strings that can be used without a namespace
- **skip_validation** (*bool*) – If true, don't download and cache namespaces/annotations

manager

This metadata parser's internal definition cache manager

namespace_to_term_to_encoding

A dictionary of cached {namespace keyword: {(identifier, name): encoding}}

uncachable_namespaces

A set of namespaces's URLs that can't be cached

namespace_to_pattern

A dictionary of {namespace keyword: regular expression string}

default_namespace

A set of names that can be used without a namespace

annotation_to_term

A dictionary of cached {annotation keyword: set of values}

annotation_to_pattern

A dictionary of {annotation keyword: regular expression string}

annotation_to_local

A dictionary of cached {annotation keyword: set of values}

document_metadata

A dictionary containing the document metadata

namespace_url_dict

A dictionary from {namespace keyword: BEL namespace URL}

annotation_url_dict

A dictionary from {annotation keyword: BEL annotation URL}

handle_document(*line, position, tokens*)

Handle statements like SET DOCUMENT X = "Y".

Raises InvalidMetadataException

Raises VersionFormatWarning

Return type ParseResults

raise_for_redefined_namespace(*line, position, namespace*)

Raise an exception if a namespace is already defined.

Raises RedefinedNamespaceError

Return type None

handle_namespace_url(*line, position, tokens*)

Handle statements like DEFINE NAMESPACE X AS URL "Y".

Raises RedefinedNamespaceError

Raises pybel.resources.exc.ResourceError

Return type ParseResults

ensure_resources()

Load all namespaces/annotations that have been encountered so far during parsing.

handle_namespace_pattern(*line, position, tokens*)

Handle statements like DEFINE NAMESPACE X AS PATTERN "Y".

Raises RedefinedNamespaceError

Return type ParseResults

raise_for_redefined_annotation(*line, position, annotation*)

Raise an exception if the given annotation is already defined.

Raises RedefinedAnnotationError

Return type None

handle_annotations_url(*line, position, tokens*)

Handle statements like DEFINE ANNOTATION X AS URL "Y".

Raises RedefinedAnnotationError

Return type ParseResults

handle_annotation_list(*line, position, tokens*)

Handle statements like `DEFINE ANNOTATION X AS LIST {"Y","Z", ...}`.

Raises RedefinedAnnotationError

Return type ParseResults

handle_annotation_pattern(*line, position, tokens*)

Handle statements like `DEFINE ANNOTATION X AS PATTERN "Y"`.

Raises RedefinedAnnotationError

Return type ParseResults

has_enumerated_annotation(*annotation*)

Check if this annotation is defined by an enumeration.

Return type bool

has_regex_annotation(*annotation*)

Check if this annotation is defined by a regular expression.

Return type bool

has_local_annotation(*annotation*)

Check if this annotation is defined by an locally.

Return type bool

has_annotation(*annotation*)

Check if this annotation is defined.

Return type bool

has_enumerated_namespace(*namespace*)

Check if this namespace is defined by an enumeration.

Return type bool

has_regex_namespace(*namespace*)

Check if this namespace is defined by a regular expression.

Return type bool

has_namespace(*namespace*)

Check if this namespace is defined.

Return type bool

raise_for_version(*line, position, version*)

Check that a version string is valid for BEL documents.

This means it's either in the YYYYMMDD or semantic version format.

Parameters

- **line** (*str*) – The line being parsed
- **position** (*int*) – The position in the line being parsed
- **version** (*str*) – A version string

Raises VersionFormatWarning

Return type None

25.3 Control Parser

```
class pybel.parser.parse_control.ControlParser(annotation_to_term=None,  
                                              annotation_to_pattern=None,  
                                              annotation_to_local=None, citation_clearing=True,  
                                              required_annotations=None)
```

A parser for BEL control statements.

See also:

BEL 1.0 specification on [control records](#)

Initialize the control statement parser.

Parameters

- **annotation_to_term** (*Optional[Mapping[str, Set[str]]]*) – A dictionary of {annotation: set of valid values} defined with URL for parsing
- **annotation_to_pattern** (*Optional[Mapping[str, Pattern]]*) – A dictionary of {annotation: regular expression string}
- **annotation_to_local** (*Optional[Mapping[str, Set[str]]]*) – A dictionary of {annotation: set of valid values} for parsing defined with LIST
- **citation_clearing** (*bool*) – Should SET Citation statements clear evidence and all annotations?
- **required_annotations** (*Optional[List[str]]*) – Annotations that are required

property citation_is_set: *bool*

Check if the citation is set.

Return type *bool*

has_enumerated_annotation(*annotation*)

Check if the annotation is defined as an enumeration.

Return type *bool*

has_regex_annotation(*annotation*)

Check if the annotation is defined as a regular expression.

Return type *bool*

has_local_annotation(*annotation*)

Check if the annotation is defined locally.

Return type *bool*

has_annotation(*annotation*)

Check if the annotation is defined.

Return type *bool*

raise_for_undefined_annotation(*line, position, annotation*)

Raise an exception if the annotation is not defined.

Raises UndefinedAnnotationWarning

Return type *None*

raise_for_invalid_annotation_value(*line, position, key, value*)

Raise an exception if the annotation is not defined.

Raises IllegalAnnotationValueWarning or MissingAnnotationRegexWarning

Return type None

raise_for_missing_citation(*line, position*)

Raise an exception if there is no citation present in the parser.

Raises MissingCitationException

Return type None

handle_annotation_key(*line, position, tokens*)

Handle an annotation key before parsing to validate that it's either enumerated or as a regex.

Raise MissingCitationException or UndefinedAnnotationWarning

Return type ParseResults

handle_set_statement_group(*_, __, tokens*)

Handle a SET STATEMENT_GROUP = "X" statement.

Return type ParseResults

handle_set_citation(*line, position, tokens*)

Handle a SET Citation = {"X", "Y", "Z", ...} statement.

Return type ParseResults

handle_set_evidence(*_, __, tokens*)

Handle a SET Evidence = "" statement.

Return type ParseResults

handle_set_command(*line, position, tokens*)

Handle a SET X = "Y" statement.

Return type ParseResults

handle_set_command_list(*line, position, tokens*)

Handle a SET X = {"Y", "Z", ...} statement.

Return type ParseResults

handle_unset_statement_group(*line, position, tokens*)

Unset the statement group, or raises an exception if it is not set.

Raises MissingAnnotationKeyWarning

Return type ParseResults

handle_unset_citation(*line, position, tokens*)

Unset the citation, or raise an exception if it is not set.

Raises MissingAnnotationKeyWarning

Return type ParseResults

handle_unset_evidence(*line, position, tokens*)

Unset the evidence, or throws an exception if it is not already set.

The value for tokens[EVIDENCE] corresponds to which alternate of SupportingText or Evidence was used in the BEL script.

Raises MissingAnnotationKeyWarning

Return type ParseResults

validate_unset_command(*line, position, annotation*)

Raise an exception when trying to UNSET *X* if *X* is not already set.

Raises MissingAnnotationKeyWarning

Return type None

handle_unset_command(*line, position, tokens*)

Handle an UNSET *X* statement or raises an exception if it is not already set.

Raises MissingAnnotationKeyWarning

Return type ParseResults

handle_unset_list(*line, position, tokens*)

Handle UNSET {*A*, *B*, ...} or raises an exception if any of them are not present.

Consider that all unsets are in peril if just one of them is wrong!

Raises MissingAnnotationKeyWarning

Return type ParseResults

handle_unset_all(*_, __, tokens*)

Handle an UNSET_ALL statement.

Return type ParseResults

get_annotations()

Get the current annotations.

Return type Dict[str, Any]

get_citation()

Get the citation dictionary.

Return type Optional[CitationDict]

get_missing_required_annotations()

Return missing required annotations.

Return type List[str]

clear_citation()

Clear the citation and if citation clearing is enabled, clear the evidence and annotations.

Return type None

clear()

Clear the statement_group, citation, evidence, and annotations.

Return type None

25.4 Concept Parser

```
class pybel.parser.parse_concept.ConceptParser(namespace_to_term_to_encoding=None,  
                                                namespace_to_pattern=None,  
                                                default_namespace=None, allow_naked_names=False,  
                                                skip_validation=False, ensure_go=True)
```

A parser for concepts in the form of namespace:name or namespace:identifier!name.

Can be made more lenient when given a default namespace or enabling the use of naked names.

Initialize the concept parser.

Parameters

- **namespace_to_term_to_encoding** (`Optional[Mapping[str, Mapping[Tuple[Optional[str], str], str]]]`) – A dictionary of {namespace: {(identifier, name): encoding}}
- **namespace_to_pattern** (`Optional[Mapping[str, Pattern]]`) – A dictionary of {namespace: regular expression string} to compile
- **default_namespace** (`Optional[Set[str]]`) – A set of strings that can be used without a namespace
- **allow_naked_names** (`bool`) – If true, turn off naked namespace failures

has_enumerated_namespace(*namespace*)

Check that the namespace has been defined by an enumeration.

Return type `bool`

has_regex_namespace(*namespace*)

Check that the namespace has been defined by a regular expression.

Return type `bool`

raise_for_missing_namespace(*line, position, namespace, name*)

Raise an exception if the namespace is not defined.

Return type `None`

raise_for_missing_name(*line, position, namespace, name*)

Raise an exception if the namespace is not defined or if it does not validate the given name.

Return type `None`

handle_identifier_fqualified(*line, position, tokens*)

Handle parsing a qualified OBO-style identifier.

Return type `ParseResults`

handle_identifier_qualified(*line, position, tokens*)

Handle parsing a qualified identifier.

Return type `ParseResults`

handle_namespace_default(*line, position, tokens*)

Handle parsing an identifier for the default namespace.

Return type `ParseResults`

static handle_namespace_lenient(*line, position, tokens*)

Handle parsing an identifier for names missing a namespace that are outside the default namespace.

Return type `ParseResults`

handle_namespace_invalid(*line, position, tokens*)

Raise an exception when parsing a name missing a namespace.

Return type `None`

25.5 Sub-Parsers

Parsers for modifications to abundances.

INTERNAL DOMAIN SPECIFIC LANGUAGE

PyBEL implements an internal domain-specific language (DSL).

This enables you to write BEL using Python scripts. Even better, you can programmatically generate BEL using Python. See the Bio2BEL [paper](#) and [repository](#) for many examples.

Internally, the BEL parser converts BEL script into the BEL DSL then adds it to a BEL graph object. When you iterate through the `pybel.BELGraph`, the nodes are instances of subclasses of `pybel.dsl.BaseEntity`.

26.1 Primitives

class `pybel.dsl.Entity(*, namespace, name=None, identifier=None)`

Represents a named entity with a namespace and name/identifier.

Create a dictionary representing a reference to an entity.

Parameters

- **namespace** (`str`) – The namespace to which the entity belongs
- **name** (`Optional[str]`) – The name of the entity
- **identifier** (`Optional[str]`) – The identifier of the entity in the namespace

class `pybel.dsl.BaseEntity`

This is the superclass for all BEL terms.

A BEL term has three properties:

1. It has a type. Subclasses of this function should set the class variable `function`.
2. It can be converted to BEL. Note, this is an abstract class, so all sub-classes must implement this functionality in `as_bel()`.
3. It can be hashed, based on the BEL conversion

class `pybel.dsl.BaseAbundance(namespace, name=None, identifier=None, xrefs=None)`

The superclass for all named BEL terms.

A named BEL term has:

1. A type (taken care of by being a subclass of `BaseEntity`)
2. A named `Entity`. Though this doesn't directly inherit from `Entity`, it creates one internally using the namespace, identifier, and name. Ideally, both the identifier and name are given. If one is missing, it can be looked up with `pybel.grounding.ground()`
3. An optional list of xrefs, corresponding to the whole entity, not just the namespace/name. For example, the BEL term `p(HGNC:APP, frag(672_713))` could xref CHEBI:64647.

Build an abundance from a function, namespace, and a name and/or identifier.

Parameters

- **namespace** (`str`) – The name of the namespace
- **name** (`Optional[str]`) – The name of this abundance
- **identifier** (`Optional[str]`) – The database identifier for this abundance
- **xrefs** (`Optional[List[Entity]]`) – Alternate identifiers for the entity

class `pybel.dsl.ListAbundance`(*members*)

The superclass for all BEL terms defined by lists, as opposed to by names like in [BaseAbundance](#).

Build a list abundance node.

Parameters **members** (`Union[BaseAbundance, Iterable[BaseAbundance]]`) – A list of PyBEL node data dictionaries

26.2 Named Entities

class `pybel.dsl.Abundance`(*namespace, name=None, identifier=None, xrefs=None*)

Builds an abundance node.

```
>>> from pybel.dsl import Abundance
>>> Abundance(namespace='CHEBI', name='water')
```

Build an abundance from a function, namespace, and a name and/or identifier.

Parameters

- **namespace** (`str`) – The name of the namespace
- **name** (`Optional[str]`) – The name of this abundance
- **identifier** (`Optional[str]`) – The database identifier for this abundance
- **xrefs** (`Optional[List[Entity]]`) – Alternate identifiers for the entity

class `pybel.dsl.BiologicalProcess`(*namespace, name=None, identifier=None, xrefs=None*)

Builds a biological process node.

```
>>> from pybel.dsl import BiologicalProcess
>>> BiologicalProcess(namespace='GO', name='apoptosis')
```

Build an abundance from a function, namespace, and a name and/or identifier.

Parameters

- **namespace** (`str`) – The name of the namespace
- **name** (`Optional[str]`) – The name of this abundance
- **identifier** (`Optional[str]`) – The database identifier for this abundance
- **xrefs** (`Optional[List[Entity]]`) – Alternate identifiers for the entity

class `pybel.dsl.Pathology`(*namespace, name=None, identifier=None, xrefs=None*)

Build a pathology node.

```
>>> from pybel.dsl import Pathology
>>> Pathology(namespace='DO', name='Alzheimer Disease')
```

Build an abundance from a function, namespace, and a name and/or identifier.

Parameters

- **namespace** (*str*) – The name of the namespace
- **name** (*Optional[str]*) – The name of this abundance
- **identifier** (*Optional[str]*) – The database identifier for this abundance
- **xrefs** (*Optional[List[Entity]]*) – Alternate identifiers for the entity

class pybel.dsl.**Population**(*namespace, name=None, identifier=None, xrefs=None*)
Builds a population node.

```
>>> from pybel.dsl import Population
>>> Population(namespace='uberon', name='blood')
```

Build an abundance from a function, namespace, and a name and/or identifier.

Parameters

- **namespace** (*str*) – The name of the namespace
- **name** (*Optional[str]*) – The name of this abundance
- **identifier** (*Optional[str]*) – The database identifier for this abundance
- **xrefs** (*Optional[List[Entity]]*) – Alternate identifiers for the entity

26.3 Central Dogma

class pybel.dsl.**CentralDogma**(*namespace, name=None, identifier=None, xrefs=None, variants=None*)
The base class for “central dogma” abundances (i.e., genes, miRNAs, RNAs, and proteins).

Build a node for a gene, RNA, miRNA, or protein.

Parameters

- **namespace** (*str*) – The name of the database used to identify this entity
- **name** (*Optional[str]*) – The database’s preferred name or label for this entity
- **identifier** (*Optional[str]*) – The database’s identifier for this entity
- **xrefs** (*Optional[List[Entity]]*) – Alternative database cross references
- **variants** (*Union[None, Variant, Iterable[Variant]]*) – An optional variant or list of variants

class pybel.dsl.**Gene**(*namespace, name=None, identifier=None, xrefs=None, variants=None*)
Builds a gene node.

Build a node for a gene, RNA, miRNA, or protein.

Parameters

- **namespace** (*str*) – The name of the database used to identify this entity
- **name** (*Optional[str]*) – The database’s preferred name or label for this entity

- **identifier** (`Optional[str]`) – The database’s identifier for this entity
- **xrefs** (`Optional[List[Entity]]`) – Alternative database cross references
- **variants** (`Union[None, Variant, Iterable[Variant]]`) – An optional variant or list of variants

class `pybel.dsl.Transcribable(namespace, name=None, identifier=None, xrefs=None, variants=None)`

A base class for RNA and micro-RNA to share getting of their corresponding genes.

Build a node for a gene, RNA, miRNA, or protein.

Parameters

- **namespace** (`str`) – The name of the database used to identify this entity
- **name** (`Optional[str]`) – The database’s preferred name or label for this entity
- **identifier** (`Optional[str]`) – The database’s identifier for this entity
- **xrefs** (`Optional[List[Entity]]`) – Alternative database cross references
- **variants** (`Union[None, Variant, Iterable[Variant]]`) – An optional variant or list of variants

class `pybel.dsl.Rna(namespace, name=None, identifier=None, xrefs=None, variants=None)`

Builds an RNA node.

Example: AKT1 protein coding gene’s RNA:

```
>>> from pybel.dsl import Rna
>>> Rna(namespace='HGNC', name='AKT1', identifier='391')
```

Non-coding RNAs can also be encoded such as U85:

```
>>> from pybel.dsl import Rna
>>> Rna(namespace='SNORNABASE', identifier='SR0000073')
```

Build a node for a gene, RNA, miRNA, or protein.

Parameters

- **namespace** (`str`) – The name of the database used to identify this entity
- **name** (`Optional[str]`) – The database’s preferred name or label for this entity
- **identifier** (`Optional[str]`) – The database’s identifier for this entity
- **xrefs** (`Optional[List[Entity]]`) – Alternative database cross references
- **variants** (`Union[None, Variant, Iterable[Variant]]`) – An optional variant or list of variants

class `pybel.dsl.MicroRna(namespace, name=None, identifier=None, xrefs=None, variants=None)`

Represents an micro-RNA.

Human miRNA’s are listed on HUGO’s [MicroRNAs \(MIR\)](#) gene family.

MIR1-1 from [HGNC](#):

```
>>> from pybel.dsl import MicroRna
>>> MicroRna(namespace='HGNC', name='MIR1-1', identifier='31499')
```

MIR1-1 from [miRBase](#):

```
>>> from pybel.dsl import MicroRna
>>> MicroRna(namespace='MIRBASE', identifier='MI0000651')
```

MIR1-1 from Entrez Gene

```
>>> from pybel.dsl import MicroRna
>>> MicroRna(namespace='ENTREZ', identifier='406904')
```

Build a node for a gene, RNA, miRNA, or protein.

Parameters

- **namespace** (*str*) – The name of the database used to identify this entity
- **name** (*Optional[str]*) – The database’s preferred name or label for this entity
- **identifier** (*Optional[str]*) – The database’s identifier for this entity
- **xrefs** (*Optional[List[Entity]]*) – Alternative database cross references
- **variants** (*Union[None, Variant, Iterable[Variant]]*) – An optional variant or list of variants

class `pybel.dsl.Protein(namespace, name=None, identifier=None, xrefs=None, variants=None)`

Builds a protein node.

Example: AKT

```
>>> from pybel.dsl import Protein
>>> Protein(namespace='HGNC', name='AKT1')
```

Example: AKT with optionally included HGNC database identifier

```
>>> from pybel.dsl import Protein
>>> Protein(namespace='HGNC', name='AKT1', identifier='391')
```

Example: AKT with phosphorylation

```
>>> from pybel.dsl import Protein, ProteinModification
>>> Protein(namespace='HGNC', name='AKT', variants=[ProteinModification('Ph', code=
↳ 'Thr', position=308)])
```

Build a node for a gene, RNA, miRNA, or protein.

Parameters

- **namespace** (*str*) – The name of the database used to identify this entity
- **name** (*Optional[str]*) – The database’s preferred name or label for this entity
- **identifier** (*Optional[str]*) – The database’s identifier for this entity
- **xrefs** (*Optional[List[Entity]]*) – Alternative database cross references
- **variants** (*Union[None, Variant, Iterable[Variant]]*) – An optional variant or list of variants

26.3.1 Variants

class `pybel.dsl.Variant(kind)`

The superclass for variant dictionaries.

Build the variant data dictionary.

Parameters `kind` (`str`) – The kind of variant

class `pybel.dsl.ProteinModification(name, code=None, position=None, namespace=None, identifier=None, xrefs=None)`

Build a protein modification variant dictionary.

Build a protein modification variant data dictionary.

Parameters

- **name** (`str`) – The name of the modification
- **code** (`Optional[str]`) – The three letter amino acid code for the affected residue. Capital first letter.
- **position** (`Optional[int]`) – The position of the affected residue
- **namespace** (`Optional[str]`) – The namespace to which the name of this modification belongs
- **identifier** (`Optional[str]`) – The identifier of the name of the modification
- **xrefs** (`Optional[List[Entity]]`) – Alternative database xrefs

Either the name or the identifier must be used. If the namespace is omitted, it is assumed that a name is specified from the BEL default namespace.

Example from BEL default namespace:

```
>>> from pybel.dsl import ProteinModification
>>> ProteinModification('Ph', code='Thr', position=308)
```

Example from custom namespace:

```
>>> from pybel.dsl import ProteinModification
>>> ProteinModification(name='protein phosphorylation', namespace='G0', code='Thr',
↪ position=308)
```

Example from custom namespace additionally qualified with identifier:

```
>>> from pybel.dsl import ProteinModification
>>> ProteinModification(name='protein phosphorylation', namespace='G0',
>>> identifier='0006468', code='Thr', position=308)
```

class `pybel.dsl.GeneModification(name, namespace=None, identifier=None, xrefs=None)`

Build a gene modification variant dictionary.

Build a protein modification variant data dictionary.

Parameters

- **name** (`str`) – The name of the modification
- **namespace** (`Optional[str]`) – The namespace to which the name of this modification belongs

- **identifier** (*Optional*[*str*]) – The identifier of the name of the modification
- **xrefs** (*Optional*[*List*[*Entity*]]) – Alternative database xrefs

Either the name or the identifier must be used. If the namespace is omitted, it is assumed that a name is specified from the BEL default namespace.

Example from BEL default namespace:

```
>>> from pybel.dsl import GeneModification
>>> GeneModification(name='Me')
```

Example from custom namespace:

```
>>> from pybel.dsl import GeneModification
>>> GeneModification(name='DNA methylation', namespace='GO', identifier='0006306')
```

class `pybel.dsl.Hgvs`(*variant*)

Builds a HGVS variant dictionary.

Build an HGVS variant data dictionary.

Parameters **variant** (*str*) – The HGVS variant string

```
>>> from pybel.dsl import Protein, Hgvs
>>> Protein(namespace='HGNC', name='AKT1', variants=[Hgvs('p.Ala127Tyr')])
```

class `pybel.dsl.HgvsReference`

Represents the “reference” variant in HGVS.

Build an HGVS variant data dictionary.

Parameters **variant** – The HGVS variant string

```
>>> from pybel.dsl import Protein, Hgvs
>>> Protein(namespace='HGNC', name='AKT1', variants=[Hgvs('p.Ala127Tyr')])
```

class `pybel.dsl.HgvsUnspecified`

Represents an unspecified variant in HGVS.

Build an HGVS variant data dictionary.

Parameters **variant** – The HGVS variant string

```
>>> from pybel.dsl import Protein, Hgvs
>>> Protein(namespace='HGNC', name='AKT1', variants=[Hgvs('p.Ala127Tyr')])
```

class `pybel.dsl.ProteinSubstitution`(*from_aa*, *position*, *to_aa*)

A protein substitution variant.

Build an HGVS variant data dictionary for the given protein substitution.

Parameters

- **from_aa** (*str*) – The 3-letter amino acid code of the original residue
- **position** (*int*) – The position of the residue
- **to_aa** (*str*) – The 3-letter amino acid code of the new residue

```
>>> from pybel.dsl import Protein, ProteinSubstitution
>>> Protein(namespace='HGNC', name='AKT1', variants=[ProteinSubstitution('Ala', 127,
↪ 'Tyr')])
```

class pybel.dsl.Fragment(*start=None, stop=None, description=None*)

Represent the information about a protein fragment.

Build a protein fragment data dictionary.

Parameters

- **start** (*Union[None, int, str]*) – The starting position
- **stop** (*Union[None, int, str]*) – The stopping position
- **description** (*Optional[str]*) – An optional description

Example of specified fragment:

```
>>> from pybel.dsl import Protein, Fragment
>>> Protein(name='APP', namespace='HGNC', variants=[Fragment(start=672, stop=713)])
```

Example of unspecified fragment:

```
>>> from pybel.dsl import Protein, Fragment
>>> Protein(name='APP', namespace='HGNC', variants=[Fragment()])
```

26.4 Fusions

class pybel.dsl.FusionBase(*partner_5p, partner_3p, range_5p=None, range_3p=None*)

The superclass for building fusion node data dictionaries.

Build a fusion node.

Parameters

- **partner_5p** (*CentralDogma*) – A PyBEL node for the 5-prime partner
- **partner_3p** (*CentralDogma*) – A PyBEL node for the 3-prime partner
- **range_5p** (*Optional[FusionRangeBase]*) – A fusion range for the 5-prime partner
- **range_3p** (*Optional[FusionRangeBase]*) – A fusion range for the 3-prime partner

class pybel.dsl.GeneFusion(*partner_5p, partner_3p, range_5p=None, range_3p=None*)

Builds a gene fusion node.

Example, using fusion ranges with the ‘c’ qualifier

```
>>> from pybel.dsl import GeneFusion, Gene
>>> GeneFusion(
>>> ... partner_5p=Gene(namespace='HGNC', name='TPRSS2'),
>>> ... range_5p=EnumeratedFusionRange('c', 1, 79),
>>> ... partner_3p=Gene(namespace='HGNC', name='ERG'),
>>> ... range_3p=EnumeratedFusionRange('c', 312, 5034)
>>> )
```

Example with missing fusion ranges:


```
>>> from pybel.dsl import GeneFusion, Gene
>>> GeneFusion(
>>> ... partner_5p=Gene(namespace='HGNC', name='TMPRSS2'),
>>> ... partner_3p=Gene(namespace='HGNC', name='ERG'),
>>> )
```

Build a fusion node.

Parameters

- **partner_5p** (*CentralDogma*) – A PyBEL node for the 5-prime partner
- **partner_3p** (*CentralDogma*) – A PyBEL node for the 3-prime partner
- **range_5p** (*Optional[FusionRangeBase]*) – A fusion range for the 5-prime partner
- **range_3p** (*Optional[FusionRangeBase]*) – A fusion range for the 3-prime partner

class pybel.dsl.**RnaFusion**(*partner_5p, partner_3p, range_5p=None, range_3p=None*)

Builds an RNA fusion node.

Example, with fusion ranges using the ‘r’ qualifier:

```
>>> from pybel.dsl import RnaFusion, Rna
>>> RnaFusion(
>>> ... partner_5p=Rna(namespace='HGNC', name='TMPRSS2'),
>>> ... range_5p=EnumeratedFusionRange('r', 1, 79),
>>> ... partner_3p=Rna(namespace='HGNC', name='ERG'),
>>> ... range_3p=EnumeratedFusionRange('r', 312, 5034)
>>> )
```

Example with missing fusion ranges:

```
>>> from pybel.dsl import RnaFusion, Rna
>>> RnaFusion(
>>> ... partner_5p=Rna(namespace='HGNC', name='TMPRSS2'),
>>> ... partner_3p=Rna(namespace='HGNC', name='ERG'),
>>> )
```

Build a fusion node.

Parameters

- **partner_5p** (*CentralDogma*) – A PyBEL node for the 5-prime partner
- **partner_3p** (*CentralDogma*) – A PyBEL node for the 3-prime partner
- **range_5p** (*Optional[FusionRangeBase]*) – A fusion range for the 5-prime partner
- **range_3p** (*Optional[FusionRangeBase]*) – A fusion range for the 3-prime partner

class pybel.dsl.**ProteinFusion**(*partner_5p, partner_3p, range_5p=None, range_3p=None*)

Builds a protein fusion node.

Build a fusion node.

Parameters

- **partner_5p** (*CentralDogma*) – A PyBEL node for the 5-prime partner
- **partner_3p** (*CentralDogma*) – A PyBEL node for the 3-prime partner
- **range_5p** (*Optional[FusionRangeBase]*) – A fusion range for the 5-prime partner

- **range_3p** (Optional[FusionRangeBase]) – A fusion range for the 3-prime partner

26.4.1 Fusion Ranges

class pybel.dsl.FusionRangeBase

The superclass for fusion range data dictionaries.

class pybel.dsl.EnumeratedFusionRange(*reference, start, stop*)

Represents an enumerated fusion range.

Build an enumerated fusion range.

Parameters

- **reference** (*str*) – The reference code
- **or str start** (*int*) – The start position, either specified by its integer position, or ‘?’
- **or str stop** (*int*) – The stop position, either specified by its integer position, ‘?’, or ‘*’

Example fully specified RNA fusion range:

```
>>> EnumeratedFusionRange('r', 1, 79)
```

class pybel.dsl.MissingFusionRange

Represents a fusion range with no defined start or end.

Build a missing fusion range.

26.4.2 List Abundances

class pybel.dsl.ComplexAbundance(*members, namespace=None, name=None, identifier=None, xrefs=None*)

Build a complex abundance node with the optional ability to specify a name.

Build a complex list node.

Parameters

- **members** (Iterable[BaseAbundance]) – A list of PyBEL node data dictionaries
- **namespace** (Optional[str]) – The namespace from which the name originates
- **name** (Optional[str]) – The name of the complex
- **identifier** (Optional[str]) – The identifier in the namespace in which the name originates
- **xrefs** (Optional[List[Entity]]) – Alternate identifiers for the entity if it is named

class pybel.dsl.CompositeAbundance(*members*)

Build a composite abundance node.

This node is effectively the “AND” inside BEL, which can help represent when two things need to be true at the same time. For example, in COVID 19, if both the NF-KB and IL6-STAT complex are present, then acute respiratory distress syndrome happens.

```
>>> from pybel.dsl import CompositeAbundance, ComplexAbundance, Protein, \
↳ NamedComplexAbundance
>>> CompositeAbundance([
...     NamedComplexAbundance('fplx', 'nfkb'),
```

(continues on next page)

(continued from previous page)

```

...     ComplexAbundance([
...         Protein('hgnc', identifier='6018', name='IL6'),
...         Protein('hgnc', identifier='11364', name='STAT3'),
...     ]),
... ])
```

Build a list abundance node.

Parameters **members** (`Union[BaseAbundance, Iterable[BaseAbundance]]`) – A list of PyBEL node data dictionaries

class `pybel.dsl.Reaction`(*reactants, products, namespace=None, name=None, identifier=None, xrefs=None*)
Build a reaction node.

Build a reaction node.

Parameters

- **reactants** (`Union[BaseAbundance, Iterable[BaseAbundance]]`) – A list of PyBEL node data dictionaries representing the reactants
- **products** (`Union[BaseAbundance, Iterable[BaseAbundance]]`) – A list of PyBEL node data dictionaries representing the products
- **namespace** (`Optional[str]`) – The namespace from which the name originates
- **name** (`Optional[str]`) – The name of the complex
- **identifier** (`Optional[str]`) – The identifier in the namespace in which the name originates
- **xrefs** (`Optional[List[Entity]]`) – Alternate identifiers for the entity if it is named

```

>>> from pybel.dsl import Reaction, Protein, Abundance
>>> Reaction([Protein(namespace='HGNC', name='KNG1')], [Abundance(namespace='CHEBI',
↪ name='bradykinin')])
```

26.5 Utilities

The following functions are useful to build DSL objects from dictionaries:

`pybel.tokens.parse_result_to_dsl(tokens)`
Convert a ParseResult to a PyBEL DSL object.

Return type `BaseEntity`

LOGGING MESSAGES

This module contains base exceptions that are shared through the package.

A message for “General Parser Failure” is displayed when a problem was caused due to an unforeseen error. The line number and original statement are printed for the user to debug.

exception `pybel.exceptions.PyBELWarning`

The base class for warnings during compilation from which PyBEL can recover.

exception `pybel.exceptions.BELParserWarning(line_number, line, position, *args)`

The base PyBEL parser exception, which holds the line and position where a parsing problem occurred.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.BELSyntaxError(line_number, line, position, *args)`

For general syntax errors.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.InconsistentDefinitionError(line_number, line, position, definition)`

Base PyBEL error for redefinition.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.RedefinedNamespaceError(line_number, line, position, definition)`

Raised when a namespace is redefined.

Initialize the BEL parser warning.

Parameters

- **line_number** (*int*) – The line number on which this warning occurred
- **line** (*str*) – The content of the line
- **position** (*int*) – The position within the line where the warning occurred

exception `pybel.exceptions.RedefinedAnnotationError(line_number, line, position, definition)`
Raised when an annotation is redefined.

Initialize the BEL parser warning.

Parameters

- **line_number** (*int*) – The line number on which this warning occurred
- **line** (*str*) – The content of the line
- **position** (*int*) – The position within the line where the warning occurred

exception `pybel.exceptions.NameWarning(line_number, line, position, name, *args)`
The base class for errors related to nomenclature.

Build a warning wrapping a given name.

exception `pybel.exceptions.NakedNameWarning(line_number, line, position, name, *args)`
Raised when there is an identifier without a namespace. Enable lenient mode to suppress.

Build a warning wrapping a given name.

exception `pybel.exceptions.MissingDefaultNameWarning(line_number, line, position, name, *args)`
Raised if reference to value not in default namespace.

Build a warning wrapping a given name.

exception `pybel.exceptions.NamespaceIdentifierWarning(line_number, line, position, namespace, name)`

The base class for warnings related to namespace:name identifiers.

Initialize the namespace identifier warning.

Parameters

- **line_number** (*int*) – The line number of the line that caused the exception
- **line** (*str*) – The line that caused the exception
- **position** (*int*) – The line's position of the exception
- **namespace** (*str*) – The namespace of the identifier
- **name** (*str*) – The name of the identifier

exception `pybel.exceptions.UndefinedNamespaceWarning(line_number, line, position, namespace, name)`
Raised if reference made to undefined namespace.

Initialize the namespace identifier warning.

Parameters

- **line_number** (*int*) – The line number of the line that caused the exception
- **line** (*str*) – The line that caused the exception
- **position** (*int*) – The line's position of the exception
- **namespace** (*str*) – The namespace of the identifier

- **name** (*str*) – The name of the identifier

exception `pybel.exceptions.MissingNamespaceNameWarning`(*line_number*, *line*, *position*, *namespace*, *name*)

Raised if reference to value not in namespace.

Initialize the namespace identifier warning.

Parameters

- **line_number** (*int*) – The line number of the line that caused the exception
- **line** (*str*) – The line that caused the exception
- **position** (*int*) – The line's position of the exception
- **namespace** (*str*) – The namespace of the identifier
- **name** (*str*) – The name of the identifier

exception `pybel.exceptions.MissingNamespaceRegexWarning`(*line_number*, *line*, *position*, *namespace*, *name*)

Raised if reference not matching regex.

Initialize the namespace identifier warning.

Parameters

- **line_number** (*int*) – The line number of the line that caused the exception
- **line** (*str*) – The line that caused the exception
- **position** (*int*) – The line's position of the exception
- **namespace** (*str*) – The namespace of the identifier
- **name** (*str*) – The name of the identifier

exception `pybel.exceptions.AnnotationWarning`(*line_number*, *line*, *position*, *annotation*, *args)

Base exception for annotation warnings.

Build an AnnotationWarning.

Parameters

- **line_number** (*int*) – The line number on which the warning occurred
- **line** (*str*) – The line on which the warning occurred
- **position** (*int*) – The position in the line that caused the warning
- **annotation** (*str*) – The annotation name that caused the warning

exception `pybel.exceptions.UndefinedAnnotationWarning`(*line_number*, *line*, *position*, *annotation*, *args)

Raised when an undefined annotation is used.

Build an AnnotationWarning.

Parameters

- **line_number** (*int*) – The line number on which the warning occurred
- **line** (*str*) – The line on which the warning occurred
- **position** (*int*) – The position in the line that caused the warning
- **annotation** (*str*) – The annotation name that caused the warning

exception `pybel.exceptions.MissingAnnotationKeyWarning`(*line_number*, *line*, *position*, *annotation*,
**args*)

Raised when trying to unset an annotation that is not set.

Build an AnnotationWarning.

Parameters

- **line_number** (*int*) – The line number on which the warning occurred
- **line** (*str*) – The line on which the warning occurred
- **position** (*int*) – The position in the line that caused the warning
- **annotation** (*str*) – The annotation name that caused the warning

exception `pybel.exceptions.AnnotationIdentifierWarning`(*line_number*, *line*, *position*, *annotation*,
value)

Base exception for annotation:value pairs.

Build an AnnotationWarning.

Parameters

- **line_number** (*int*) – The line number on which the warning occurred
- **line** (*str*) – The line on which the warning occurred
- **position** (*int*) – The position in the line that caused the warning
- **annotation** (*str*) – The annotation name that caused the warning

exception `pybel.exceptions.IllegalAnnotationValueWarning`(*line_number*, *line*, *position*, *annotation*,
value)

Raised when an annotation has a value that does not belong to the original set of valid annotation values.

Build an AnnotationWarning.

Parameters

- **line_number** (*int*) – The line number on which the warning occurred
- **line** (*str*) – The line on which the warning occurred
- **position** (*int*) – The position in the line that caused the warning
- **annotation** (*str*) – The annotation name that caused the warning

exception `pybel.exceptions.MissingAnnotationRegexWarning`(*line_number*, *line*, *position*, *annotation*,
value)

Raised if annotation doesn't match regex.

Build an AnnotationWarning.

Parameters

- **line_number** (*int*) – The line number on which the warning occurred
- **line** (*str*) – The line on which the warning occurred
- **position** (*int*) – The position in the line that caused the warning
- **annotation** (*str*) – The annotation name that caused the warning

exception `pybel.exceptions.VersionFormatWarning`(*line_number*, *line*, *position*, *version_string*)

Raised if the version string doesn't adhere to semantic versioning or YYYYMMDD format.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.MetadataException(line_number, line, position, *args)`

Base exception for issues with document metadata.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.MalformedMetadataException(line_number, line, position, *args)`

Raised when an invalid metadata line is encountered.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.InvalidMetadataException(line_number, line, position, key, value)`

Raised when an incorrect document metadata key is used.

Hint: Valid document metadata keys are:

- Authors
- ContactInfo
- Copyright
- Description
- Disclaimer
- Licenses
- Name
- Version

See also:

BEL specification on the [properties section](#)

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.MissingMetadataException(line_number, line, position, key)`

Raised when a BEL Script is missing critical metadata.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

static make(key)

Build an instance of this class with auto-filled dummy values.

Unlike normal classes, polymorphism on `__init__` can't be used for exceptions when pickling/unpickling.

exception `pybel.exceptions.InvalidCitationLengthException(line_number, line, position, *args)`

Base exception raised when the format for a citation is wrong.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.CitationTooShortException(line_number, line, position, *args)`

Raised when a citation does not have the minimum of {type, name, reference}.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.CitationTooLongException(line_number, line, position, *args)`

Raised when a citation has more than the allowed entries, {type, name, reference, date, authors, comments}.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.MissingCitationException(line_number, line, position, *args)`

Raised when trying to parse a BEL statement, but no citation is currently set.

This might be due to a previous error in the formatting of a citation.

Though it's not a best practice, some BEL curators set other annotations before the citation. If this is the case in your BEL document, and you're *absolutely* sure that all UNSET statements are correctly written, you can use `citation_clearing=True` as a keyword argument in any of the IO functions in `pybel.from_lines()`, `pybel.from_url()`, or `pybel.from_path()`.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.MissingSupportWarning`(*line_number, line, position, *args*)

Raised when trying to parse a BEL statement, but no evidence is currently set.

All BEL statements must be qualified with evidence.

If your data is serialized from a database and provenance information is not readily accessible, consider referencing the publication for the database, or a url pointing to the data from either a programmatically or human-readable endpoint.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.MissingAnnotationWarning`(*line_number, line, position, required_annotations*)

Raised when trying to parse a BEL statement and a required annotation is not present.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.InvalidCitationType`(*line_number, line, position, citation_type*)

Raised when a citation is set with an incorrect type.

Hint: Valid citation types include:

- Book
 - PubMed
 - Journal
 - Online Resource
 - URL
 - DOI
 - Other
-

See also:

OpenBEL wiki on [citations](#)

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.InvalidPubMedIdentifierWarning(line_number, line, position, reference)`

Raised when a citation is set whose type is PubMed but whose database identifier is not a valid integer.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.MalformedTranslocationWarning(line_number, line, position, tokens)`

Raised when there is a translocation statement without location information.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.PlaceholderAminoAcidWarning(line_number, line, position, code)`

Raised when an invalid amino acid code is given.

One example might be the usage of X, which is a colloquial signifier for a truncation in a given position. Text mining efforts for knowledge extraction make this mistake often. X might also signify a placeholder amino acid.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.NestedRelationWarning(line_number, line, position, *args)`

Raised when encountering a nested statement.

See our the docs for an explanation of why we explicitly do not support nested statements.

Initialize the BEL parser warning.

Parameters

- **line_number** (`int`) – The line number on which this warning occurred
- **line** (`str`) – The content of the line
- **position** (`int`) – The position within the line where the warning occurred

exception `pybel.exceptions.InvalidEntity(line_number, line, position, namespace, name)`

Raised when using a non-entity name for a name.

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

exception `pybel.exceptions.InvalidFunctionSemantic(line_number, line, position, func, namespace, name, allowed_functions)`

Raised when an invalid function is used for a given node.

For example, an HGNC symbol for a protein-coding gene YFG cannot be referenced as an miRNA with `m(HGNC:YFG)`

Initialize the BEL parser warning.

Parameters

- **line_number** – The line number on which this warning occurred
- **line** – The content of the line
- **position** – The position within the line where the warning occurred

REFERENCES

If you find PyBEL useful for your work, please consider citing¹:

28.1 Related Publications

We have used PyBEL in several other projects and publications. Below is a sample:

- Domingo-Fernández, D., Mubeen, S., Marin-Llao, J., Hoyt, C. T., & Hofmann-Apitius, M. (2018). PathMe: Merging and exploring mechanistic pathway knowledge. *bioRxiv*, 451625.
- Domingo-Fernández, D., Hoyt, C. T., Alvarez, C. B., Marin-Llao, J., Hofmann-Apitius, M. (2018). ComPath: an ecosystem for exploring, analyzing, and curating mappings across pathway databases. *Npj Systems Biology and Applications*, 5(1), 3. <https://doi.org/10.1038/s41540-018-0078-8>
- Hoyt, C. T., et al. (2018). A systematic approach for identifying shared mechanisms in epilepsy and its comorbidities. *Database*, 2018(1). <https://doi.org/10.1093/database/bay050>
- Hoyt, C. T., et al. (2019). Re-curation and Rational Enrichment of Knowledge Graphs in Biological Expression Language. *BioRxiv*, 536409. <https://doi.org/10.1101/536409>
- Hoyt, C. T., Domingo-Fernández, D., & Hofmann-Apitius, M. (2018). BEL Commons: an environment for exploration and analysis of networks encoded in Biological Expression Language. *Database*, 2018(3), 1–11. <https://doi.org/10.1093/database/bay126>
- Ali, M., et al. (2019). BioKEEN: A library for learning and evaluating biological knowledge graph embeddings. *Bioinformatics*, btz117. <https://doi.org/10.1093/bioinformatics/btz117>

28.2 Software using PyBEL

- <https://github.com/cthoit/bel-repository>
- <https://github.com/bio2bel>
- <https://github.com/sorgerlab/indra>
- <https://github.com/smartDataAnalytics/biokeen>

¹ Hoyt, C. T., *et al.* (2017). PyBEL: a Computational Framework for Biological Expression Language. *Bioinformatics*, 34(December), 1–2.

28.3 BEL Content

- <https://github.com/neurommsig/neurommsig-knowledge>
- <https://github.com/pharmacome/knowledge>

CURRENT ISSUES

29.1 Speed

Speed is still an issue, because documents above 100K lines still take a couple minutes to run. This issue is exacerbated by (optionally) logging output to the console, which can make it more than 3x or 4x as slow.

29.2 Namespaces

The default namespaces from OpenBEL do not follow a standard file format. They are similar to INI config files, but do not use consistent delimiters. Also, many of the namespaces don't respect that the delimiter should not be used in the namespace names. There are also lots of names with strange characters, which may have been caused by copying from a data source that had specific escape characters without proper care.

29.3 Testing

Testing was very difficult because the example documents on the OpenBEL website had many semantic errors, such as using names and annotation values that were not defined within their respective namespace and annotation definition files. They also contained syntax errors like naked names, which are not only syntactically incorrect, but lead to bad science; and improper usage of activities, like illegally nesting an activity within a composite statement.

TECHNOLOGY

This page is meant to describe the development stack for PyBEL, and should be a useful introduction for contributors.

30.1 Versioning

PyBEL is versioned on GitHub so changes in its code can be tracked over time and to make use of the variety of software development plugins. Code is produced following the [Git Flow](#) philosophy, which means that new features are coded in branches off of the development branch and merged after they are triaged. Finally, develop is merged into master for releases. If there are bugs in releases that need to be fixed quickly, “hot fix” branches from master can be made, then merged back to master and develop after fixing the problem.

30.2 Testing in PyBEL

PyBEL is written with extensive unit testing and integration testing. Whenever possible, test-driven development is practiced. This means that new ideas for functions and features are encoded as blank classes/functions and directly writing tests for the desired output. After tests have been written that define how the code should work, the implementation can be written.

Test-driven development requires us to think about design before making quick and dirty implementations. This results in better code. Additionally, thorough testing suites make it possible to catch when changes break existing functionality.

Tests are written with the standard `unittest` library.

30.2.1 Unit Testing

Unit tests check that the functionality of the different parts of PyBEL work independently.

An example unit test can be found in `tests.test_parse_bel.TestAbundance.test_short_abundance`. It ensures that the parser is able to handle a given string describing the abundance of a chemical/other entity in BEL. It tests that the parser produces the correct output, that the BEL statement is converted to the correct internal representation. In this example, this is a tuple describing the abundance of oxygen atoms. Finally, it tests that this representation is added as a node in the underlying BEL graph with the appropriate attributes added.

30.2.2 Integration Testing

Integration tests are more high level, and ensure that the software accomplishes more complicated goals by using many components. An example integration test is found in `tests.test_import.TestImport.test_from_fileURL`. This test ensures that a BEL script can be read and results in a `NetworkX` object that contains all of the information described in the script

30.2.3 Tox

While IDEs like PyCharm provide excellent testing tools, they are not programmatic. `Tox` is python package that provides a CLI interface to run automated testing procedures (as well as other build functions, that aren't important to explain here). In PyBEL, it is used to run the unit tests in the `tests` folder with the `pytest` harness. It also runs `check-manifest`, builds the documentation with `sphinx`, and computes the code coverage of the tests. The entire procedure is defined in `tox.ini`. Tox also allows test to be done on many different versions of Python.

30.2.4 Continuous Integration

Continuous integration is a philosophy of automatically testing code as it changes. PyBEL makes use of the Travis CI server to perform testing because of its tight integration with GitHub. Travis automatically installs git hooks inside GitHub so it knows when a new commit is made. Upon each commit, Travis downloads the newest commit from GitHub and runs the tests configured in the `.travis.yml` file in the top level of the PyBEL repository. This file effectively instructs the Travis CI server to run Tox. It also allows for the modification of the environment variables. This is used in PyBEL to test many different versions of python.

30.2.5 Code Coverage

After building, Travis sends code coverage results to [covercov.io](https://coveralls.io). This site helps visualize untested code and track the improvement of testing coverage over time. It also integrates with GitHub to show which feature branches are inadequately tested. In development of PyBEL, inadequately tested code is not allowed to be merged into develop.

30.2.6 Versioning

PyBEL uses semantic versioning. In general, the project's version string will has a suffix `-dev` like in `0.3.4-dev` throughout the development cycle. After code is merged from feature branches to develop and it is time to deploy, this suffix is removed and develop branch is merged into master.

The version string appears in multiple places throughout the project, so `BumpVersion` is used to automate the updating of these version strings. See `.bumpversion.cfg` for more information.

30.3 Deployment

PyBEL is also distributed through PyPI (pronounced Py-Pee-Eye). Travis CI has a wonderful integration with PyPI, so any time a tag is made on the master branch (and also assuming the tests pass), a new distribution is packed and sent to PyPI. Refer to the “deploy” section at the bottom of the `.travis.yml` file for more information, or the Travis CI [PyPI deployment documentation](#). As a side note, Travis CI has an encryption tool so the password for the PyPI account can be displayed publicly on GitHub. Travis decrypts it before performing the upload to PyPI.

30.3.1 Steps

1. `bumpversion release` on development branch
2. Push to git
3. After tests pass, merge develop in to master
4. After tests pass, create a tag on GitHub with the same name as the version number (on master)
5. Travis will automatically deploy to PyPI after tests pass. After checking deployment has been successful, switch to develop and `bumpversion patch`

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

[Tarca2009] Tarca, A. L., *et al* (2009). [A novel signaling pathway impact analysis](#). *Bioinformatics*, 25(1), 75–82.

PYTHON MODULE INDEX

p

- `pybel.constants`, 139
- `pybel.dsl`, 163
- `pybel.examples`, 39
 - `pybel.examples.braf_example`, 40
 - `pybel.examples.egf_example`, 39
 - `pybel.examples.sialic_acid_example`, 40
 - `pybel.examples.statin_example`, 41
 - `pybel.examples.tloc_example`, 41
- `pybel.exceptions`, 175
- `pybel.io`, 89
 - `pybel.io.aws`, 108
 - `pybel.io.bel_commons_client`, 107
 - `pybel.io.biodati_client`, 109
 - `pybel.io.cx`, 95
 - `pybel.io.emmaa`, 112
 - `pybel.io.extras`, 115
 - `pybel.io.fraunhofer_orientdb`, 111
 - `pybel.io.gpickle`, 92
 - `pybel.io.graphdati`, 98
 - `pybel.io.graphml`, 114
 - `pybel.io.hetionet`, 91
 - `pybel.io.hipathia`, 103
 - `pybel.io.indra`, 100
 - `pybel.io.jgif`, 96
 - `pybel.io.jupyter`, 101
 - `pybel.io.neo4j`, 113
 - `pybel.io.nodelink`, 93
 - `pybel.io.pykeen`, 105
 - `pybel.io.pynpa`, 102
 - `pybel.io.sbel`, 94
 - `pybel.io.spia`, 104
 - `pybel.io.triples`, 106
 - `pybel.io.umbrella_nodelink`, 114
- `pybel.language`, 146
- `pybel.manager.database_io`, 113
- `pybel.manager.models`, 123
- `pybel.parser.modifiers`, 162
 - `pybel.parser.modifiers.fragment`, 30
 - `pybel.parser.modifiers.fusion`, 31
 - `pybel.parser.modifiers.gene_modification`, 27
 - `pybel.parser.modifiers.gene_substitution`, 26
 - `pybel.parser.modifiers.location`, 36
 - `pybel.parser.modifiers.protein_modification`, 28
 - `pybel.parser.modifiers.protein_substitution`, 27
 - `pybel.parser.modifiers.truncation`, 29
 - `pybel.parser.modifiers.variant`, 25
- `pybel.struct`, 11
 - `pybel.struct.filters`, 43
 - `pybel.struct.getters`, 70
 - `pybel.struct.grouping`, 53
 - `pybel.struct.mutation.collapse`, 73
 - `pybel.struct.mutation.deletion`, 75
 - `pybel.struct.mutation.expansion`, 77
 - `pybel.struct.mutation.induction`, 79
 - `pybel.struct.mutation.induction_expansion`, 83
 - `pybel.struct.mutation.inference`, 85
 - `pybel.struct.mutation.metadata`, 87
 - `pybel.struct.mutation.utils`, 71
 - `pybel.struct.pipeline.decorators`, 59
 - `pybel.struct.pipeline.exc`, 60
 - `pybel.struct.query`, 61
 - `pybel.struct.summary`, 65

Symbols

- `__add__()` (*pybel.BELGraph* method), 12
- `__and__()` (*pybel.BELGraph* method), 13
- `__iadd__()` (*pybel.BELGraph* method), 12
- `__iand__()` (*pybel.BELGraph* method), 13
- `--allow-naked-names`
 - `pybel-compile` command line option, 132
- `--bel <bel>`
 - `pybel-serialize` command line option, 137
- `--connection <connection>`
 - `pybel` command line option, 131
 - `pybel-neo` command line option, 136
- `--debug`
 - `pybel-manage-examples` command line option, 134
- `--disallow-nested`
 - `pybel-compile` command line option, 132
- `--disallow-unqualified-translocations`
 - `pybel-compile` command line option, 132
- `--edgelist <edgelist>`
 - `pybel-serialize` command line option, 137
- `--graphml <graphml>`
 - `pybel-serialize` command line option, 137
- `--gsea <gsea>`
 - `pybel-serialize` command line option, 137
- `--host <host>`
 - `pybel-machine` command line option, 133
 - `pybel-upload` command line option, 138
- `--limit <limit>`
 - `pybel-manage-edges-ls` command line option, 134
- `--local`
 - `pybel-machine` command line option, 133
- `--namespace-id <namespace_id>`
 - `pybel-manage-namespaces-ls` command line option, 135
- `--network-id <network_id>`
 - `pybel-manage-networks-drop` command line option, 135
- `--no-citation-clearing`
 - `pybel-compile` command line option, 132
- `--no-identifier-validation`
 - `pybel-compile` command line option, 132
- `--nodelink <nodelink>`
 - `pybel-serialize` command line option, 137
- `--offset <offset>`
 - `pybel-manage-edges-ls` command line option, 134
- `--password <password>`
 - `pybel-neo` command line option, 136
 - `pybel-upload` command line option, 138
- `--pprint`
 - `pybel-parse` command line option, 137
- `--required-annotations`
 - `<required_annotations>`
 - `pybel-compile` command line option, 132
- `--sif <sif>`
 - `pybel-serialize` command line option, 137
- `--skip-tqdm`
 - `pybel-compile` command line option, 132
- `--tsv <tsv>`
 - `pybel-serialize` command line option, 137
- `--upgrade-urls`
 - `pybel-compile` command line option, 132
- `--url <url>`
 - `pybel-manage-namespaces-ls` command line option, 135
- `--user <user>`
 - `pybel-upload` command line option, 138
- `--verbose`
 - `pybel-compile` command line option, 132
- `--version`
 - `pybel` command line option, 131
- `--yes`
 - `pybel-manage-drop` command line option, 133
 - `pybel-manage-networks-drop` command line option, 135
- `-c`
 - `pybel` command line option, 131
- `-i`
 - `pybel-manage-namespaces-ls` command line option, 135
- `-n`

pybel-manage-networks-drop command line option, 135

-r
pybel-compile command line option, 132

-u
pybel-manage-namespaces-ls command line option, 135

-v
pybel-compile command line option, 132
pybel-manage-examples command line option, 134

-y
pybel-manage-networks-drop command line option, 135

A

Abundance (*class in pybel.dsl*), 164

ABUNDANCE (*in module pybel.constants*), 141

abundance_labels (*in module pybel.language*), 147

abundance_list (*pybel.parser.parse_bel.BELParser attribute*), 153

abundance_sbo_mapping (*in module pybel.language*), 147

activity (*pybel.parser.parse_bel.BELParser attribute*), 152

activity_labels (*in module pybel.language*), 146

activity_mapping (*in module pybel.language*), 146

add_activates() (*pybel.BELGraph method*), 20

add_annotation_value() (*in module pybel.struct.mutation.metadata*), 87

add_association() (*pybel.BELGraph method*), 19

add_binds() (*pybel.BELGraph method*), 18

add_causes_no_change() (*pybel.BELGraph method*), 20

add_correlation() (*pybel.BELGraph method*), 19

add_decreases() (*pybel.BELGraph method*), 19

add_dephosphorylates() (*pybel.BELGraph method*), 21

add_directly_decreases() (*pybel.BELGraph method*), 19

add_directly_dephosphorylates() (*pybel.BELGraph method*), 21

add_directly_increases() (*pybel.BELGraph method*), 19

add_directly_phosphorylates() (*pybel.BELGraph method*), 20

add_directly_regulates() (*pybel.BELGraph method*), 19

add_equivalence() (*pybel.BELGraph method*), 17

add_has_product() (*pybel.BELGraph method*), 18

add_has_reactant() (*pybel.BELGraph method*), 18

add_has_variant() (*pybel.BELGraph method*), 18

add_increases() (*pybel.BELGraph method*), 18

add_inhibits() (*pybel.BELGraph method*), 20

add_is_a() (*pybel.BELGraph method*), 18

add_negative_correlation() (*pybel.BELGraph method*), 20

add_no_correlation() (*pybel.BELGraph method*), 20

add_node_from_data() (*pybel.BELGraph method*), 21

add_orthology() (*pybel.BELGraph method*), 17

add_part_of() (*pybel.BELGraph method*), 18

add_phosphorylates() (*pybel.BELGraph method*), 20

add_positive_correlation() (*pybel.BELGraph method*), 20

add_qualified_edge() (*pybel.BELGraph method*), 18

add_reaction() (*pybel.BELGraph method*), 21

add_regulates() (*pybel.BELGraph method*), 19

add_transcription() (*pybel.BELGraph method*), 17

add_transitivity() (*pybel.BELGraph method*), 17

add_translation() (*pybel.BELGraph method*), 17

add_unqualified_edge() (*pybel.BELGraph method*), 17

AGENTS
pybel-machine command line option, 133

ANALOGOUS_TO (*in module pybel.constants*), 143

and_edge_predicates() (*in module pybel.struct.filters*), 43

annotation() (*pybel.struct.graph.InduceDispatch method*), 23

annotation_curie (*pybel.BELGraph property*), 16

annotation_list (*pybel.BELGraph property*), 16

annotation_miriam (*pybel.BELGraph property*), 16

annotation_pattern (*pybel.BELGraph property*), 16

annotation_to_local (*pybel.parser.parse_metadata.MetadataParser attribute*), 156

annotation_to_pattern (*pybel.parser.parse_metadata.MetadataParser attribute*), 156

annotation_to_term (*pybel.parser.parse_metadata.MetadataParser attribute*), 156

annotation_url (*pybel.BELGraph property*), 16

annotation_url_dict (*pybel.parser.parse_metadata.MetadataParser attribute*), 156

AnnotationIdentifierWarning, 178

ANNOTATIONS (*in module pybel.constants*), 141

annotations() (*pybel.struct.graph.CountDispatch method*), 23

AnnotationWarning, 177

append() (*pybel.Pipeline method*), 58

append_annotation() (*pybel.struct.query.Seeding method*), 63

append_induction() (*pybel.struct.query.Seeding method*), 62

append_neighbors() (*pybel.struct.query.Seeding method*), 63

- [append_network\(\)](#) (*pybel.struct.query.Query method*), [61](#)
[append_pipeline\(\)](#) (*pybel.struct.query.Query method*), [62](#)
[append_sample\(\)](#) (*pybel.struct.query.Seeding method*), [63](#)
[append_seeding_annotation\(\)](#) (*pybel.struct.query.Query method*), [61](#)
[append_seeding_induction\(\)](#) (*pybel.struct.query.Query method*), [61](#)
[append_seeding_neighbors\(\)](#) (*pybel.struct.query.Query method*), [61](#)
[append_seeding_sample\(\)](#) (*pybel.struct.query.Query method*), [61](#)
[article_type](#) (*pybel.manager.models.Citation attribute*), [126](#)
[as_bel\(\)](#) (*pybel.manager.models.Network method*), [125](#)
[as_bel\(\)](#) (*pybel.manager.models.Node method*), [126](#)
[ASSOCIATION](#) (*in module pybel.constants*), [143](#)
[Author](#) (*class in pybel.manager.models*), [126](#)
[author](#) (*pybel.manager.models.Namespace attribute*), [124](#)
[authors](#) (*pybel.BELGraph property*), [14](#)
[authors](#) (*pybel.manager.models.Network attribute*), [125](#)
[authors\(\)](#) (*pybel.struct.graph.CountDispatch method*), [23](#)
- ## B
- [Base](#) (*class in pybel.manager.models*), [123](#)
[base](#) (*pybel.manager.BaseManager attribute*), [117](#)
[BaseAbundance](#) (*class in pybel.dsl*), [163](#)
[BaseEntity](#) (*class in pybel.dsl*), [163](#)
[BaseManager](#) (*class in pybel.manager*), [117](#)
[bel](#) (*pybel.manager.models.Edge attribute*), [127](#)
[bel](#) (*pybel.manager.models.Node attribute*), [126](#)
[bel_contains\(\)](#) (*pybel.manager.models.Node class method*), [126](#)
[BELGraph](#) (*class in pybel*), [12](#)
[belns_encodings](#) (*in module pybel.constants*), [145](#)
[BELParser](#) (*class in pybel.parser.parse_bel*), [151](#)
[BELParserWarning](#), [175](#)
[BELSyntaxError](#), [175](#)
[bind\(\)](#) (*pybel.manager.BaseManager method*), [117](#)
[BINDS](#) (*in module pybel.constants*), [142](#)
[biological_process](#) (*pybel.parser.parse_bel.BELParser attribute*), [152](#)
[BiologicalProcess](#) (*class in pybel.dsl*), [164](#)
[BIOMARKER_FOR](#) (*in module pybel.constants*), [143](#)
[BIOPROCESS](#) (*in module pybel.constants*), [141](#)
[blob](#) (*pybel.manager.models.Network attribute*), [125](#)
[build_annotation_dict_all_filter\(\)](#) (*in module pybel.struct.filters*), [43](#)
[build_annotation_dict_any_filter\(\)](#) (*in module pybel.struct.filters*), [43](#)
[build_author_inclusion_filter\(\)](#) (*in module pybel.struct.filters*), [44](#)
[build_downstream_edge_predicate\(\)](#) (*in module pybel.struct.filters*), [44](#)
[build_node_data_search\(\)](#) (*in module pybel.struct.filters*), [46](#)
[build_node_graph_data_search\(\)](#) (*in module pybel.struct.filters*), [47](#)
[build_node_key_search\(\)](#) (*in module pybel.struct.filters*), [47](#)
[build_node_name_search\(\)](#) (*in module pybel.struct.filters*), [47](#)
[build_pmid_exclusion_filter\(\)](#) (*in module pybel.struct.filters*), [43](#)
[build_pmid_inclusion_filter\(\)](#) (*in module pybel.struct.filters*), [44](#)
[build_relation_predicate\(\)](#) (*in module pybel.struct.filters*), [44](#)
[build_upstream_edge_predicate\(\)](#) (*in module pybel.struct.filters*), [43](#)
- ## C
- [calculate_error_by_annotation\(\)](#) (*in module pybel.struct.summary*), [67](#)
[calculate_incorrect_name_dict\(\)](#) (*in module pybel.struct.summary*), [66](#)
[CAUSAL_APOLAR_RELATIONS](#) (*in module pybel.constants*), [143](#)
[CAUSAL_DECREASE_RELATIONS](#) (*in module pybel.constants*), [143](#)
[CAUSAL_INCREASE_RELATIONS](#) (*in module pybel.constants*), [143](#)
[CAUSAL_POLAR_RELATIONS](#) (*in module pybel.constants*), [143](#)
[CAUSAL_RELATIONS](#) (*in module pybel.constants*), [143](#)
[CAUSES_NO_CHANGE](#) (*in module pybel.constants*), [142](#)
[CentralDogma](#) (*class in pybel.dsl*), [165](#)
[check_function_semantics\(\)](#) (*pybel.parser.parse_bel.BELParser method*), [153](#)
[child\(\)](#) (*pybel.BELGraph method*), [13](#)
[Citation](#) (*class in pybel.manager.models*), [126](#)
[CITATION](#) (*in module pybel.constants*), [141](#)
[CITATION_ARTICLE_TYPE](#) (*in module pybel.constants*), [139](#)
[CITATION_AUTHORS](#) (*in module pybel.constants*), [139](#)
[CITATION_DATE](#) (*in module pybel.constants*), [139](#)
[citation_dict\(\)](#) (*in module pybel.language*), [150](#)
[CITATION_FIRST_AUTHOR](#) (*in module pybel.constants*), [139](#)
[citation_is_set](#) (*pybel.parser.parse_control.ControlParser*

- property), 158
- CITATION_ISSUE (in module *pybel.constants*), 139
- CITATION_JOURNAL (in module *pybel.constants*), 139
- CITATION_LAST_AUTHOR (in module *pybel.constants*), 139
- CITATION_PAGES (in module *pybel.constants*), 139
- CITATION_VOLUME (in module *pybel.constants*), 139
- CitationDict (class in *pybel.language*), 150
- citations() (*pybel.struct.graph.CountDispatch* method), 23
- citations() (*pybel.struct.graph.SummarizeDispatch* method), 24
- CitationTooLongException, 180
- CitationTooShortException, 180
- clear() (*pybel.parser.parse_bel.BELParser* method), 153
- clear() (*pybel.parser.parse_control.ControlParser* method), 160
- clear_citation() (*pybel.parser.parse_control.ControlParser* method), 160
- collapse_all_variants() (in module *pybel.struct.mutation.collapse*), 73
- collapse_nodes() (in module *pybel.struct.mutation.collapse*), 73
- collapse_pair() (in module *pybel.struct.mutation.collapse*), 73
- collapse_to_genes() (in module *pybel.struct.mutation.collapse*), 73
- compartment_mapping (in module *pybel.language*), 147
- COMPLEX (in module *pybel.constants*), 141
- complex_singleton (*pybel.parser.parse_bel.BELParser* attribute), 152
- ComplexAbundance (class in *pybel.dsl*), 172
- COMPOSITE (in module *pybel.constants*), 141
- composite_abundance (*pybel.parser.parse_bel.BELParser* attribute), 152
- CompositeAbundance (class in *pybel.dsl*), 172
- concatenate_node_predicates() (in module *pybel.struct.filters*), 50
- CONCEPT (in module *pybel.constants*), 140
- ConceptParser (class in *pybel.parser.parse_concept*), 160
- contact (*pybel.BELGraph* property), 14
- contact (*pybel.manager.models.Namespace* attribute), 124
- contact (*pybel.manager.models.Network* attribute), 125
- ControlParser (class in *pybel.parser.parse_control*), 158
- copyright (*pybel.BELGraph* property), 15
- copyright (*pybel.manager.models.Network* attribute), 125
- CORRELATION (in module *pybel.constants*), 142
- CORRELATIVE_RELATIONS (in module *pybel.constants*), 143
- count (*pybel.BELGraph* property), 13
- count_annotations() (in module *pybel.struct.summary*), 66
- count_edges() (*pybel.manager.QueryManager* method), 120
- count_error_types() (in module *pybel.struct.summary*), 66
- count_functions() (in module *pybel.struct.summary*), 67
- count_naked_names() (in module *pybel.struct.summary*), 66
- count_names_by_namespace() (in module *pybel.struct.summary*), 67
- count_namespaces() (in module *pybel.struct.summary*), 67
- count_networks() (*pybel.manager.NetworkManager* method), 119
- count_nodes() (*pybel.manager.QueryManager* method), 120
- count_passed_edge_filter() (in module *pybel.struct.filters*), 43
- count_passed_node_filter() (in module *pybel.struct.filters*), 46
- count_pathologies() (in module *pybel.struct.summary*), 69
- count_relations() (in module *pybel.struct.summary*), 65
- count_variants() (in module *pybel.struct.summary*), 69
- CountDispatch (class in *pybel.struct.graph*), 22
- create_all() (*pybel.manager.BaseManager* method), 117
- created (*pybel.manager.models.Namespace* attribute), 124
- curie (*pybel.language.Entity* property), 146
- ## D
- data (*pybel.manager.models.Edge* attribute), 128
- data (*pybel.manager.models.Node* attribute), 126
- data_missing_key_builder() (in module *pybel.struct.filters*), 46
- date (*pybel.manager.models.Citation* attribute), 127
- db (*pybel.manager.models.Citation* attribute), 126
- db_id (*pybel.manager.models.Citation* attribute), 126
- DECREASES (in module *pybel.constants*), 142
- default_namespace (*pybel.parser.parse_metadata.MetadataParser* attribute), 155
- defined_annotation_keywords (*pybel.BELGraph* property), 16

- defined_namespace_keywords (*pybel.BELGraph* property), 15
- degradation (*pybel.parser.parse_bel.BELParser* attribute), 152
- DeprecationMappingError, 60
- DESCRIPTION (*in module pybel.constants*), 140
- description (*pybel.BELGraph* property), 14
- description (*pybel.manager.models.Namespace* attribute), 123
- description (*pybel.manager.models.Network* attribute), 125
- description_contains() (*pybel.manager.models.Network* class method), 125
- dict() (*pybel.struct.graph.SummarizeDispatch* method), 24
- DIRECT_CAUSAL_RELATIONS (*in module pybel.constants*), 143
- DIRECTLY_DECREASES (*in module pybel.constants*), 142
- DIRECTLY_INCREASES (*in module pybel.constants*), 142
- DIRECTLY_REGULATES (*in module pybel.constants*), 142
- DIRTY (*in module pybel.constants*), 141
- disclaimer (*pybel.BELGraph* property), 15
- disclaimer (*pybel.manager.models.Network* attribute), 125
- DOCUMENT_KEYS (*in module pybel.constants*), 144
- document_metadata (*pybel.parser.parse_metadata.MetadataParser* attribute), 156
- domain (*pybel.manager.models.Namespace* attribute), 123
- drop_all() (*pybel.manager.BaseManager* method), 117
- drop_network() (*pybel.manager.NetworkManager* method), 119
- drop_network_by_id() (*pybel.manager.NetworkManager* method), 119
- drop_networks() (*pybel.manager.NetworkManager* method), 119
- dump() (*in module pybel*), 89
- dump() (*pybel.Pipeline* method), 58
- dump() (*pybel.struct.query.Query* method), 62
- dump() (*pybel.struct.query.Seeding* method), 63
- dumps() (*pybel.Pipeline* method), 58
- dumps() (*pybel.struct.query.Query* method), 62
- dumps() (*pybel.struct.query.Seeding* method), 63
- ## E
- Edge (*class in pybel.manager.models*), 127
- edge_has_activity() (*in module pybel.struct.filters*), 45
- edge_has_annotation() (*in module pybel.struct.filters*), 45
- edge_has_degradation() (*in module pybel.struct.filters*), 45
- edge_has_translocation() (*in module pybel.struct.filters*), 45
- edge_predicate() (*in module pybel.struct.filters*), 44
- edge_to_bel() (*pybel.BELGraph* static method), 22
- edges() (*pybel.struct.graph.SummarizeDispatch* method), 24
- encoding (*pybel.manager.models.NamespaceEntry* attribute), 124
- enrich_protein_and_rna_origins() (*in module pybel.struct.mutation.inference*), 85
- enrich_proteins_with_rnas() (*in module pybel.struct.mutation.inference*), 85
- enrich_rnas_with_genes() (*in module pybel.struct.mutation.inference*), 85
- ensure_node() (*pybel.parser.parse_bel.BELParser* method), 154
- ensure_resources() (*pybel.parser.parse_metadata.MetadataParser* method), 156
- Entity (*class in pybel.dsl*), 163
- Entity (*class in pybel.language*), 146
- EnumeratedFusionRange (*class in pybel.dsl*), 172
- EQUIVALENT_TO (*in module pybel.constants*), 143
- error_types() (*pybel.struct.graph.CountDispatch* method), 23
- Evidence (*class in pybel.manager.models*), 127
- EVIDENCE (*in module pybel.constants*), 141
- expand (*pybel.BELGraph* property), 14
- expand_all_node_neighborhoods() (*in module pybel.struct.mutation.expansion*), 77
- expand_by_edge_filter() (*in module pybel.struct.mutation.utils*), 71
- expand_downstream_causal() (*in module pybel.struct.mutation.expansion*), 78
- expand_internal() (*in module pybel.struct.mutation.expansion*), 78
- expand_node_neighborhood() (*in module pybel.struct.mutation.expansion*), 77
- expand_node_predecessors() (*in module pybel.struct.mutation.expansion*), 77
- expand_node_successors() (*in module pybel.struct.mutation.expansion*), 77
- expand_nodes_neighborhoods() (*in module pybel.struct.mutation.expansion*), 77
- expand_upstream_causal() (*in module pybel.struct.mutation.expansion*), 78
- ExpandDispatch (*class in pybel.struct.graph*), 24
- extend() (*pybel.Pipeline* method), 58
- ## F
- false_edge_predicate() (*in module pybel.struct.filters*), 44

- false_node_predicate() (in module *pybel.struct.filters*), 51
 filter_edges() (in module *pybel.struct.filters*), 43
 filter_nodes() (in module *pybel.struct.filters*), 46
 first_id (*pybel.manager.models.Citation* attribute), 127
 Fragment (class in *pybel.dsl*), 170
 FRAGMENT (in module *pybel.constants*), 140
 FRAGMENT_DESCRIPTION (in module *pybel.constants*), 145
 FRAGMENT_MISSING (in module *pybel.constants*), 145
 FRAGMENT_START (in module *pybel.constants*), 145
 FRAGMENT_STOP (in module *pybel.constants*), 145
 from_bel_commons() (in module *pybel*), 107
 from_bel_script() (in module *pybel*), 91
 from_bel_script_url() (in module *pybel*), 91
 from_biodati() (in module *pybel*), 110
 from_biopax() (in module *pybel*), 101
 from_bytes() (in module *pybel*), 92
 from_bytes_gz() (in module *pybel*), 92
 from_cbn_jgif() (in module *pybel*), 98
 from_cbn_jgif_file() (in module *pybel*), 98
 from_cx() (in module *pybel*), 95
 from_cx_file() (in module *pybel*), 95
 from_cx_gz() (in module *pybel*), 96
 from_cx_jsons() (in module *pybel*), 95
 from_database() (in module *pybel*), 113
 from_emmaa() (in module *pybel*), 112
 from_fraunhofer_orientdb() (in module *pybel*), 111
 from_functions() (*pybel.Pipeline* static method), 57
 from_graphdati() (in module *pybel*), 98
 from_graphdati_file() (in module *pybel*), 99
 from_graphdati_gz() (in module *pybel*), 99
 from_graphdati_jsons() (in module *pybel*), 99
 from_hetionet_file() (in module *pybel*), 91
 from_hetionet_gz() (in module *pybel*), 91
 from_hetionet_json() (in module *pybel*), 91
 from_hipathia_dfs() (in module *pybel*), 104
 from_hipathia_paths() (in module *pybel*), 104
 from_indra_statements() (in module *pybel*), 100
 from_indra_statements_json() (in module *pybel*), 100
 from_indra_statements_json_file() (in module *pybel*), 101
 from_jgif() (in module *pybel*), 96
 from_jgif_file() (in module *pybel*), 97
 from_jgif_gz() (in module *pybel*), 97
 from_jgif_jsons() (in module *pybel*), 97
 from_json() (*pybel.Pipeline* static method), 58
 from_json() (*pybel.struct.query.Query* static method), 62
 from_json() (*pybel.struct.query.Seeding* static method), 63
 from_nodelink() (in module *pybel*), 93
 from_nodelink_file() (in module *pybel*), 93
 from_nodelink_gz() (in module *pybel*), 94
 from_nodelink_jsons() (in module *pybel*), 93
 from_pickle() (in module *pybel*), 92
 from_pickle_gz() (in module *pybel*), 93
 from_s3() (in module *pybel*), 109
 from_sbhel() (in module *pybel*), 94
 from_sbhel_file() (in module *pybel*), 94
 from_sbhel_gz() (in module *pybel*), 94
 FUNCTION (in module *pybel.constants*), 139
 function_exclusion_filter_builder() (in module *pybel.struct.filters*), 46
 function_inclusion_filter_builder() (in module *pybel.struct.filters*), 46
 functions() (*pybel.struct.graph.CountDispatch* method), 22
 fusion (*pybel.parser.parse_bel.BELParser* attribute), 152
 FusionBase (class in *pybel.dsl*), 170
 FusionRangeBase (class in *pybel.dsl*), 172
- ## G
- Gene (class in *pybel.dsl*), 165
 GENE (in module *pybel.constants*), 141
 gene (*pybel.parser.parse_bel.BELParser* attribute), 152
 GeneFusion (class in *pybel.dsl*), 170
 GeneModification (class in *pybel.dsl*), 168
 general_abundance (*pybel.parser.parse_bel.BELParser* attribute), 152
 get_annotation_values() (in module *pybel.struct.summary*), 65
 get_annotation_values_by_annotation() (in module *pybel.struct.summary*), 65
 get_annotations() (in module *pybel.struct.summary*), 65
 get_annotations() (*pybel.parser.parse_bel.BELParser* method), 153
 get_annotations() (*pybel.parser.parse_control.ControlParser* method), 160
 get_annotations_json() (*pybel.manager.models.Edge* method), 128
 get_authors() (*pybel.BELGraph* method), 17
 get_cache_connection() (in module *pybel.constants*), 139
 get_causal_subgraph() (in module *pybel.struct.mutation.induction*), 82
 get_citation() (*pybel.parser.parse_control.ControlParser* method), 160
 get_downstream_causal_subgraph() (in module *pybel.struct.mutation.induction*), 81

- `get_edge_annotations()` (*pybel.BELGraph* method), 21
`get_edge_citation()` (*pybel.BELGraph* method), 21
`get_edge_evidence()` (*pybel.BELGraph* method), 21
`get_edges_with_annotation()` (*pybel.manager.QueryManager* method), 121
`get_edges_with_citation()` (*pybel.manager.QueryManager* method), 120
`get_edges_with_citations()` (*pybel.manager.QueryManager* method), 121
`get_equivalent_nodes()` (*pybel.BELGraph* method), 22
`get_functions()` (in module *pybel.struct.summary*), 67
`get_graph_by_id()` (*pybel.manager.NetworkManager* method), 120
`get_graph_by_ids()` (*pybel.manager.NetworkManager* method), 120
`get_graph_by_most_recent()` (*pybel.manager.NetworkManager* method), 120
`get_graph_by_name_version()` (*pybel.manager.NetworkManager* method), 119
`get_graph_with_random_edges()` (in module *pybel.struct.mutation.induction*), 81
`get_graphs_by_ids()` (*pybel.manager.NetworkManager* method), 120
`get_hetionet()` (in module *pybel*), 91
`get_largest_component()` (in module *pybel.struct.mutation.induction*), 82
`get_metaedge_to_key()` (in module *pybel.struct.summary*), 66
`get_missing_required_annotations()` (*pybel.parser.parse_control.ControlParser* method), 160
`get_most_recent_network_by_name()` (*pybel.manager.NetworkManager* method), 120
`get_multi_causal_downstream()` (in module *pybel.struct.mutation.induction_expansion*), 83
`get_multi_causal_upstream()` (in module *pybel.struct.mutation.induction_expansion*), 83
`get_naked_names()` (in module *pybel.struct.summary*), 66
`get_names()` (in module *pybel.struct.summary*), 67
`get_names_by_namespace()` (in module *pybel.struct.summary*), 68
`get_namespaces()` (in module *pybel.struct.summary*), 67
`get_network_by_id()` (*pybel.manager.NetworkManager* method), 120
`get_network_by_name_version()` (*pybel.manager.NetworkManager* method), 119
`get_network_versions()` (*pybel.manager.NetworkManager* method), 119
`get_networks_by_ids()` (*pybel.manager.NetworkManager* method), 120
`get_networks_by_name()` (*pybel.manager.NetworkManager* method), 119
`get_nodes()` (in module *pybel.struct.filters*), 46
`get_nodes_by_function()` (in module *pybel.struct.filters*), 46
`get_nodes_by_namespace()` (in module *pybel.struct.filters*), 46
`get_nodes_in_all_shortest_paths()` (in module *pybel.struct.mutation.induction*), 80
`get_pmc_identifiers()` (in module *pybel.struct.summary*), 70
`get_pubmed_identifiers()` (in module *pybel.struct.summary*), 69
`get_random_node()` (in module *pybel.struct.mutation.induction*), 81
`get_random_path()` (in module *pybel.struct.mutation.induction*), 80
`get_random_subgraph()` (in module *pybel.struct.mutation.induction*), 81
`get_subgraph()` (in module *pybel.struct.query*), 64
`get_subgraph_by_all_shortest_paths()` (in module *pybel.struct.mutation.induction*), 80
`get_subgraph_by_annotation_value()` (in module *pybel.struct.mutation.induction*), 79
`get_subgraph_by_annotations()` (in module *pybel.struct.mutation.induction*), 79
`get_subgraph_by_authors()` (in module *pybel.struct.mutation.induction*), 79
`get_subgraph_by_edge_filter()` (in module *pybel.struct.mutation.induction*), 81
`get_subgraph_by_induction()` (in module *pybel.struct.mutation.induction*), 82
`get_subgraph_by_neighborhood()` (in module *pybel.struct.mutation.induction*), 80
`get_subgraph_by_node_filter()` (in module *pybel.struct.mutation.induction*), 82
`get_subgraph_by_pubmed()` (in module *pybel.struct.mutation.induction*), 79
`get_subgraph_by_second_neighbors()` (in module *pybel.struct.mutation.induction_expansion*), 83
`get_subgraphs_by_annotation()` (in module *pybel.struct.grouping*), 53
`get_subgraphs_by_citation()` (in module *py-*

<i>bel.struct.grouping</i>), 53		<i>handle_document()</i> (py-
<i>get_syntax_errors()</i> (in module <i>py-</i>		<i>bel.parser.parse_metadata.MetadataParser</i>
<i>bel.struct.summary</i>), 66		method), 156
<i>get_term_to_encodings()</i> (py-		<i>handle_has_components()</i> (py-
<i>bel.manager.models.Namespace</i> method),		<i>bel.parser.parse_bel.BELParser</i> method),
124		153
<i>get_tf_pairs()</i> (in module <i>pybel.struct.getters</i>), 70		<i>handle_has_members()</i> (py-
<i>get_top_hubs()</i> (in module <i>pybel.struct.summary</i>), 69		<i>bel.parser.parse_bel.BELParser</i> method),
<i>get_top_pathologies()</i> (in module <i>py-</i>		153
<i>bel.struct.summary</i>), 69		<i>handle_identifier_fqualified()</i> (py-
<i>get_transformation()</i> (in module <i>py-</i>		<i>bel.parser.parse_concept.ConceptParser</i>
<i>bel.struct.pipeline.decorators</i>), 59		method), 161
<i>get_triples_from_bel()</i> (in module <i>pybel.io.pykeen</i>),		<i>handle_identifier_qualified()</i> (py-
106		<i>bel.parser.parse_concept.ConceptParser</i>
<i>get_triples_from_bel_commons()</i> (in module <i>py-</i>		method), 161
<i>bel.io.pykeen</i>), 106		<i>handle_inverse_unqualified_relation()</i> (py-
<i>get_triples_from_bel_nodelink()</i> (in module <i>py-</i>		<i>bel.parser.parse_bel.BELParser</i> method),
<i>bel.io.pykeen</i>), 106		154
<i>get_triples_from_bel_pickle()</i> (in module <i>py-</i>		<i>handle_namespace_default()</i> (py-
<i>bel.io.pykeen</i>), 106		<i>bel.parser.parse_concept.ConceptParser</i>
<i>get_ungrounded_nodes()</i> (in module <i>py-</i>		method), 161
<i>bel.struct.summary</i>), 69		<i>handle_namespace_invalid()</i> (py-
<i>get_unused_annotations()</i> (in module <i>py-</i>		<i>bel.parser.parse_concept.ConceptParser</i>
<i>bel.struct.summary</i>), 66		method), 161
<i>get_unused_list_annotation_values()</i> (in module		<i>handle_namespace_lenient()</i> (py-
<i>pybel.struct.summary</i>), 66		<i>bel.parser.parse_concept.ConceptParser</i>
<i>get_unused_namespaces()</i> (in module <i>py-</i>		static method), 161
<i>bel.struct.summary</i>), 67		<i>handle_namespace_pattern()</i> (py-
<i>get_upstream_causal_subgraph()</i> (in module <i>py-</i>		<i>bel.parser.parse_metadata.MetadataParser</i>
<i>bel.struct.mutation.induction</i>), 81		method), 156
<i>GMOD</i> (in module <i>pybel.constants</i>), 140		<i>handle_namespace_url()</i> (py-
<i>gmod</i> (<i>pybel.parser.parse_bel.BELParser</i> attribute), 152		<i>bel.parser.parse_metadata.MetadataParser</i>
<i>gmod_mappings</i> (in module <i>pybel.language</i>), 150		method), 156
<i>gmod_namespace</i> (in module <i>pybel.language</i>), 150		<i>handle_nested_relation()</i> (py-
<i>GMOD_ORDER</i> (in module <i>pybel.constants</i>), 145		<i>bel.parser.parse_bel.BELParser</i> method),
<i>GRAPH_METADATA</i> (in module <i>pybel.constants</i>), 144		153
<i>ground()</i> (<i>pybel.BELGraph</i> method), 22		<i>handle_set_citation()</i> (py-
<i>GSUB_POSITION</i> (in module <i>pybel.constants</i>), 145		<i>bel.parser.parse_control.ControlParser</i>
<i>GSUB_REFERENCE</i> (in module <i>pybel.constants</i>), 145		method), 159
<i>GSUB_VARIANT</i> (in module <i>pybel.constants</i>), 145		<i>handle_set_command()</i> (py-
		<i>bel.parser.parse_control.ControlParser</i>
		method), 159
H		<i>handle_set_command_list()</i> (py-
<i>handle_annotation_key()</i> (py-		<i>bel.parser.parse_control.ControlParser</i>
<i>bel.parser.parse_control.ControlParser</i>		method), 159
method), 159		<i>handle_set_evidence()</i> (py-
<i>handle_annotation_list()</i> (py-		<i>bel.parser.parse_control.ControlParser</i>
<i>bel.parser.parse_metadata.MetadataParser</i>		method), 159
method), 157		<i>handle_set_statement_group()</i> (py-
<i>handle_annotation_pattern()</i> (py-		<i>bel.parser.parse_control.ControlParser</i>
<i>bel.parser.parse_metadata.MetadataParser</i>		method), 159
method), 157		<i>handle_term()</i> (<i>pybel.parser.parse_bel.BELParser</i>
<i>handle_annotations_url()</i> (py-		method), 153
<i>bel.parser.parse_metadata.MetadataParser</i>		<i>handle_translocation_illegal()</i> (py-
method), 156		

[bel.parser.parse_bel.BELParser](#) (method), [154](#)
[handle_unqualified_relation\(\)](#) (py-
[bel.parser.parse_bel.BELParser](#) method), [153](#)
[handle_unset_all\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [160](#)
[handle_unset_citation\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [159](#)
[handle_unset_command\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [160](#)
[handle_unset_evidence\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [159](#)
[handle_unset_list\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [160](#)
[handle_unset_statement_group\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [159](#)
[has_activity\(\)](#) (in module [pybel.struct.filters](#)), [47](#)
[has_annotation\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [158](#)
[has_annotation\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[has_authors\(\)](#) (in module [pybel.struct.filters](#)), [44](#)
[has_causal_edges\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[has_causal_in_edges\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[has_causal_out_edges\(\)](#) (in module [pybel.struct.filters](#)), [49](#)
[has_edge_citation\(\)](#) ([pybel.BELGraph](#) method), [21](#)
[has_edge_evidence\(\)](#) ([pybel.BELGraph](#) method), [21](#)
[has_edge_modifier\(\)](#) (in module [pybel.struct.filters](#)), [47](#)
[has_enumerated_annotation\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [158](#)
[has_enumerated_annotation\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[has_enumerated_namespace\(\)](#) (py-
[bel.parser.parse_concept.ConceptParser](#) method), [161](#)
[has_enumerated_namespace\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[has_fragment\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[has_gene_modification\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[has_hgvs\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[has_in_edges\(\)](#) (in module [pybel.struct.filters](#)), [49](#)
[has_local_annotation\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [158](#)
[has_local_annotation\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[has_member](#) ([pybel.parser.parse_bel.BELParser](#) attribute), [153](#)
[has_name_in\(\)](#) ([pybel.manager.models.Author](#) class method), [126](#)
[has_name_version\(\)](#) ([pybel.manager.NetworkManager](#) method), [119](#)
[has_namespace\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[has_out_edges\(\)](#) (in module [pybel.struct.filters](#)), [49](#)
[has_pathology_causal\(\)](#) (in module [pybel.struct.filters](#)), [45](#)
[has_pmc\(\)](#) (in module [pybel.struct.filters](#)), [44](#)
[has_polarity\(\)](#) (in module [pybel.struct.filters](#)), [45](#)
[HAS_PRODUCT](#) (in module [pybel.constants](#)), [142](#)
[has_protein_modification\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[has_provenance\(\)](#) (in module [pybel.struct.filters](#)), [44](#)
[has_pubmed\(\)](#) (in module [pybel.struct.filters](#)), [44](#)
[HAS_REACTANT](#) (in module [pybel.constants](#)), [142](#)
[has_regex_annotation\(\)](#) (py-
[bel.parser.parse_control.ControlParser](#) method), [158](#)
[has_regex_annotation\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[has_regex_namespace\(\)](#) (py-
[bel.parser.parse_concept.ConceptParser](#) method), [161](#)
[has_regex_namespace\(\)](#) (py-
[bel.parser.parse_metadata.MetadataParser](#) method), [157](#)
[HAS_VARIANT](#) (in module [pybel.constants](#)), [142](#)
[has_variant\(\)](#) (in module [pybel.struct.filters](#)), [48](#)
[HASH](#) (in module [pybel.constants](#)), [142](#)
[Hgvs](#) (class in [pybel.dsl](#)), [169](#)
[HGVS](#) (in module [pybel.constants](#)), [140](#)
[HgvsReference](#) (class in [pybel.dsl](#)), [169](#)
[HgvsUnspecified](#) (class in [pybel.dsl](#)), [169](#)
I
[id_in\(\)](#) ([pybel.manager.models.Network](#) class method), [125](#)
[IDENTIFIER](#) (in module [pybel.constants](#)), [140](#)
[identifier](#) ([pybel.language.Entity](#) property), [146](#)

- identifier (*pybel.manager.models.NamespaceEntry attribute*), 124
 IllegalAnnotationValueWarning, 178
 in_place_transformation() (*in module pybel.struct.pipeline.decorators*), 59
 InconsistentDefinitionError, 175
 INCREASES (*in module pybel.constants*), 142
 INDIRECT_CAUSAL_RELATIONS (*in module pybel.constants*), 143
 induce (*pybel.BELGraph property*), 14
 InduceDispatch (*class in pybel.struct.graph*), 23
 infer_child_relations() (*in module pybel.struct.mutation.inference*), 85
 insert_into_graph() (*pybel.manager.models.Edge method*), 128
 internal() (*pybel.struct.graph.ExpandDispatch method*), 24
 intersection() (*pybel.Pipeline static method*), 59
 InvalidCitationLengthException, 180
 InvalidCitationType, 181
 InvalidEntity, 182
 InvalidFunctionSemantic, 183
 InvalidMetadataException, 179
 InvalidPubMedIdentifierWarning, 182
 INVERSE_DOCUMENT_KEYS (*in module pybel.constants*), 144
 invert_edge_predicate() (*in module pybel.struct.filters*), 43
 invert_node_predicate() (*in module pybel.struct.filters*), 51
 IS_A (*in module pybel.constants*), 143
 is_associative_relation() (*in module pybel.struct.filters*), 45
 is_causal_central() (*in module pybel.struct.filters*), 49
 is_causal_relation() (*in module pybel.struct.filters*), 44
 is_causal_sink() (*in module pybel.struct.filters*), 49
 is_causal_source() (*in module pybel.struct.filters*), 49
 is_degraded() (*in module pybel.struct.filters*), 48
 is_direct_causal_relation() (*in module pybel.struct.filters*), 45
 is_enriched (*pybel.manager.models.Citation property*), 127
 is_isolated_list_abundance() (*in module pybel.struct.filters*), 48
 is_pubmed (*pybel.manager.models.Citation property*), 127
 is_translocated() (*in module pybel.struct.filters*), 48
 issue (*pybel.manager.models.Citation attribute*), 126
 iter_annotation_value_pairs() (*in module pybel.struct.summary*), 65
 iter_annotation_values() (*in module pybel.struct.summary*), 65
 iter_equivalent_nodes() (*pybel.BELGraph method*), 22
 iter_sample_metaedges() (*in module pybel.struct.summary*), 66
 iterate_entities() (*in module pybel.struct.summary*), 68
 iterate_node_entities() (*in module pybel.struct.summary*), 68
 iterate_pmc_identifiers() (*in module pybel.struct.summary*), 69
 iterate_pubmed_identifiers() (*in module pybel.struct.summary*), 69
- ## J
- journal (*pybel.manager.models.Citation attribute*), 126
- ## K
- keyword (*pybel.manager.models.Namespace attribute*), 123
 KIND (*in module pybel.constants*), 140
- ## L
- LABEL (*in module pybel.constants*), 140
 last_id (*pybel.manager.models.Citation attribute*), 127
 left_full_join() (*in module pybel.struct*), 55
 left_outer_join() (*in module pybel.struct*), 55
 license (*pybel.BELGraph property*), 15
 license (*pybel.manager.models.Namespace attribute*), 124
 licenses (*pybel.manager.models.Network attribute*), 125
 LINE (*in module pybel.constants*), 142
 list() (*pybel.struct.graph.SummarizeDispatch method*), 24
 list_networks() (*pybel.manager.NetworkManager method*), 119
 list_recent_networks() (*pybel.manager.NetworkManager method*), 119
 ListAbundance (*class in pybel.dsl*), 164
 load() (*in module pybel*), 89
 load() (*pybel.Pipeline static method*), 58
 load() (*pybel.struct.query.Query static method*), 62
 load() (*pybel.struct.query.Seeding static method*), 63
 loads() (*pybel.Pipeline static method*), 59
 loads() (*pybel.struct.query.Query static method*), 62
 loads() (*pybel.struct.query.Seeding static method*), 63
 location (*pybel.parser.parse_bel.BELParser attribute*), 152
- ## M
- make() (*pybel.exceptions.MissingMetadataException static method*), 180

- MalformedMetadataException, 179
- MalformedTranslocationWarning, 182
- Manager (class in *pybel.manager*), 117
- manager (*pybel.parser.parse_metadata.MetadataParser* attribute), 155
- md5 (*pybel.manager.models.Edge* attribute), 128
- MEMBERS (in module *pybel.constants*), 140
- METADATA_AUTHORS (in module *pybel.constants*), 144
- METADATA_CONTACT (in module *pybel.constants*), 144
- METADATA_COPYRIGHT (in module *pybel.constants*), 144
- METADATA_DESCRIPTION (in module *pybel.constants*), 144
- METADATA_DISCLAIMER (in module *pybel.constants*), 144
- METADATA_INSERT_KEYS (in module *pybel.constants*), 144
- METADATA_LICENSES (in module *pybel.constants*), 144
- METADATA_NAME (in module *pybel.constants*), 144
- METADATA_PROJECT (in module *pybel.constants*), 144
- METADATA_VERSION (in module *pybel.constants*), 144
- MetadataException, 179
- MetadataParser (class in *pybel.parser.parse_metadata*), 155
- MetaValueError, 60
- MicroRna (class in *pybel.dsl*), 166
- miriam_id (*pybel.manager.models.Namespace* attribute), 123
- MIRNA (in module *pybel.constants*), 141
- mirna (*pybel.parser.parse_bel.BELParser* attribute), 152
- MissingAnnotationKeyWarning, 177
- MissingAnnotationRegexWarning, 178
- MissingAnnotationWarning, 181
- MissingCitationException, 180
- MissingDefaultNameWarning, 176
- MissingFusionRange (class in *pybel.dsl*), 172
- MissingMetadataException, 179
- MissingNamespaceNameWarning, 177
- MissingNamespaceRegexWarning, 177
- MissingPipelineFunctionError, 60
- MissingSupportWarning, 181
- MissingUniverseError, 60
- modifications() (*pybel.struct.graph.CountDispatch* method), 23
- module
 - pybel.constants*, 139
 - pybel.dsl*, 163
 - pybel.examples*, 39
 - pybel.examples.braf_example*, 40
 - pybel.examples.egf_example*, 39
 - pybel.examples.sialic_acid_example*, 40
 - pybel.examples.statin_example*, 41
 - pybel.examples.tloc_example*, 41
 - pybel.exceptions*, 175
 - pybel.io*, 89
 - pybel.io.aws*, 108
 - pybel.io.bel_commons_client*, 107
 - pybel.io.biodati_client*, 109
 - pybel.io.cx*, 95
 - pybel.io.emmaa*, 112
 - pybel.io.extras*, 115
 - pybel.io.fraunhofer_orientdb*, 111
 - pybel.io.gpickle*, 92
 - pybel.io.graphdati*, 98
 - pybel.io.graphml*, 114
 - pybel.io.hetionet*, 91
 - pybel.io.hipathia*, 103
 - pybel.io.indra*, 100
 - pybel.io.jgif*, 96
 - pybel.io.jupyter*, 101
 - pybel.io.neo4j*, 113
 - pybel.io.nodelink*, 93
 - pybel.io.pykeen*, 105
 - pybel.io.pynpa*, 102
 - pybel.io.sbel*, 94
 - pybel.io.spia*, 104
 - pybel.io.triples*, 106
 - pybel.io.umbrella_nodelink*, 114
 - pybel.language*, 146
 - pybel.manager.database_io*, 113
 - pybel.manager.models*, 123
 - pybel.parser.modifiers*, 162
 - pybel.parser.modifiers.fragment*, 30
 - pybel.parser.modifiers.fusion*, 31
 - pybel.parser.modifiers.gene_modification*, 27
 - pybel.parser.modifiers.gene_substitution*, 26
 - pybel.parser.modifiers.location*, 36
 - pybel.parser.modifiers.protein_modification*, 28
 - pybel.parser.modifiers.protein_substitution*, 27
 - pybel.parser.modifiers.truncation*, 29
 - pybel.parser.modifiers.variant*, 25
 - pybel.struct*, 11
 - pybel.struct.filters*, 43
 - pybel.struct.getters*, 70
 - pybel.struct.grouping*, 53
 - pybel.struct.mutation.collapse*, 73
 - pybel.struct.mutation.deletion*, 75
 - pybel.struct.mutation.expansion*, 77
 - pybel.struct.mutation.induction*, 79
 - pybel.struct.mutation.induction_expansion*, 83
 - pybel.struct.mutation.inference*, 85
 - pybel.struct.mutation.metadata*, 87
 - pybel.struct.mutation.utils*, 71
 - pybel.struct.pipeline.decorators*, 59
 - pybel.struct.pipeline.exc*, 60

pybel.struct.query, 61
 pybel.struct.summary, 65
 molecular_activity (py-bel.parser.parse_bel.BELParser attribute), 152

N

NakedNameWarning, 176
 NAME (in module pybel.constants), 140
 name (pybel.BELGraph property), 14
 name (pybel.language.Entity property), 146
 name (pybel.manager.models.Namespace attribute), 123
 name (pybel.manager.models.NamespaceEntry attribute), 124
 name (pybel.manager.models.Network attribute), 125
 name_contains() (pybel.manager.models.Author class method), 126
 name_contains() (pybel.manager.models.NamespaceEntry class method), 124
 name_contains() (pybel.manager.models.Network class method), 125
 Namespace (class in pybel.manager.models), 123
 NAMESPACE (in module pybel.constants), 140
 namespace (pybel.language.Entity property), 146
 NAMESPACE_DOMAIN_TYPES (in module pybel.constants), 139
 namespace_inclusion_builder() (in module pybel.struct.filters), 47
 namespace_pattern (pybel.BELGraph property), 15
 namespace_to_pattern (py-bel.parser.parse_metadata.MetadataParser attribute), 155
 namespace_to_term_to_encoding (py-bel.parser.parse_metadata.MetadataParser attribute), 155
 namespace_url (pybel.BELGraph property), 15
 namespace_url_dict (py-bel.parser.parse_metadata.MetadataParser attribute), 156
 NamespaceEntry (class in pybel.manager.models), 124
 NamespaceIdentifierWarning, 176
 namespaces() (pybel.struct.graph.CountDispatch method), 22
 namespaces() (pybel.struct.graph.SummarizeDispatch method), 24
 NameWarning, 176
 NEGATIVE_CORRELATION (in module pybel.constants), 143
 neighborhood() (pybel.struct.graph.ExpandDispatch method), 24
 neighborhood() (pybel.struct.graph.InduceDispatch method), 23
 NestedRelationWarning, 182

Network (class in pybel.manager.models), 124
 NetworkManager (class in pybel.manager), 119
 no_causal_edges() (in module pybel.struct.filters), 50
 no_causal_in_edges() (in module pybel.struct.filters), 50
 no_causal_out_edges() (in module pybel.struct.filters), 50
 NO_CORRELATION (in module pybel.constants), 142
 no_in_edges() (in module pybel.struct.filters), 50
 no_out_edges() (in module pybel.struct.filters), 50
 Node (class in pybel.manager.models), 125
 node_has_namespace() (pybel.BELGraph method), 22
 node_is_grounded() (in module pybel.struct.summary), 68
 node_predicate() (in module pybel.struct.filters), 51
 node_to_bel() (pybel.BELGraph static method), 21
 NodeDegreeIterError, 61
 nodes() (pybel.struct.graph.SummarizeDispatch method), 24
 none_of() (in module pybel.struct.filters), 48
 not_causal_relation() (in module pybel.struct.filters), 45
 number_of_authors() (pybel.BELGraph method), 17
 number_of_citations() (pybel.BELGraph method), 17
 number_of_warnings() (pybel.BELGraph method), 16

O

obo (pybel.language.Entity property), 146
 one_of() (in module pybel.struct.filters), 48
 ORTHOLOGOUS (in module pybel.constants), 143

P

pages (pybel.manager.models.Citation attribute), 127
 parent (pybel.BELGraph attribute), 13
 parent (pybel.struct.graph.ExpandDispatch property), 24
 parse() (pybel.parser.parse_bel.BELParser method), 153
 parse_lines() (in module pybel.io.line_utils), 154
 parse_result_to_dsl() (in module pybel.tokens), 173
 part_has_modifier() (in module pybel.struct.filters), 51
 PART_OF (in module pybel.constants), 143
 PARTNER_3P (in module pybel.constants), 140
 PARTNER_5P (in module pybel.constants), 140
 PATH
 pybel-compile command line option, 132
 path
 pybel-insert command line option, 132
 pybel-neo command line option, 136
 pybel-serialize command line option, 137
 pybel-summarize command line option, 138
 pybel-upload command line option, 138

- pybel-warnings command line option, 138
- path (*pybel.BELGraph* property), 14
- pathologies() (*pybel.struct.graph.CountDispatch* method), 23
- Pathology (class in *pybel.dsl*), 164
- PATHOLOGY (in module *pybel.constants*), 141
- pathology (*pybel.parser.parse_bel.BELParser* attribute), 152
- paths() (*pybel.struct.graph.InduceDispatch* method), 23
- pattern (*pybel.manager.models.Namespace* attribute), 123
- periphery() (*pybel.struct.graph.ExpandDispatch* method), 24
- Pipeline (class in *pybel*), 57
- PipelineNameError, 60
- PlaceholderAminoAcidWarning, 182
- plot (*pybel.BELGraph* property), 14
- PlotDispatch (class in *pybel.struct.graph*), 24
- PMOD (in module *pybel.constants*), 140
- pmod (*pybel.parser.parse_bel.BELParser* attribute), 152
- PMOD_CODE (in module *pybel.constants*), 145
- pmod_legacy_labels (in module *pybel.language*), 150
- pmod_mappings (in module *pybel.language*), 148
- pmod_namespace (in module *pybel.language*), 147
- PMOD_ORDER (in module *pybel.constants*), 145
- PMOD_POSITION (in module *pybel.constants*), 145
- POLAR_RELATIONS (in module *pybel.constants*), 144
- Population (class in *pybel.dsl*), 165
- POPULATION (in module *pybel.constants*), 141
- POSITIVE_CORRELATION (in module *pybel.constants*), 143
- post_jgif() (in module *pybel*), 98
- PRODUCTS (in module *pybel.constants*), 140
- PROGONSTIC_BIOMARKER_FOR (in module *pybel.constants*), 143
- Protein (class in *pybel.dsl*), 167
- PROTEIN (in module *pybel.constants*), 141
- protein (*pybel.parser.parse_bel.BELParser* attribute), 152
- ProteinFusion (class in *pybel.dsl*), 171
- ProteinModification (class in *pybel.dsl*), 168
- ProteinSubstitution (class in *pybel.dsl*), 169
- prune_protein_rna_origins() (in module *pybel.struct.mutation.deletion*), 75
- PSUB_POSITION (in module *pybel.constants*), 145
- PSUB_REFERENCE (in module *pybel.constants*), 145
- PSUB_VARIANT (in module *pybel.constants*), 145
- pybel command line option
 - connection <connection>, 131
 - version, 131
 - c, 131
- pybel.constants
 - module, 139
- pybel.dsl
 - module, 163
- pybel.examples
 - module, 39
 - pybel.examples.braf_example
 - module, 40
 - pybel.examples.braf_example_graph (in module *pybel.examples.braf_example*), 41
 - pybel.examples.egf_example
 - module, 39
 - pybel.examples.egf_graph (in module *pybel.examples.egf_example*), 40
 - pybel.examples.ras_tloc_graph (in module *pybel.examples.tloc_example*), 41
 - pybel.examples.sialic_acid_example
 - module, 40
 - pybel.examples.sialic_acid_graph (in module *pybel.examples.sialic_acid_example*), 40
 - pybel.examples.statin_example
 - module, 41
 - pybel.examples.statin_graph (in module *pybel.examples.statin_example*), 41
 - pybel.examples.tloc_example
 - module, 41
- pybel.exceptions
 - module, 175
- pybel.io
 - module, 89
 - pybel.io.aws
 - module, 108
 - pybel.io.bel_commons_client
 - module, 107
 - pybel.io.biodati_client
 - module, 109
 - pybel.io.cx
 - module, 95
 - pybel.io.emmaa
 - module, 112
 - pybel.io.extras
 - module, 115
 - pybel.io.fraunhofer_orientdb
 - module, 111
 - pybel.io.gpickle
 - module, 92
 - pybel.io.graphdati
 - module, 98
 - pybel.io.graphml
 - module, 114
 - pybel.io.hetionet
 - module, 91
 - pybel.io.hipathia
 - module, 103
 - pybel.io.indra
 - module, 100
 - pybel.io.jgif

- module, 96
- pybel.io.jupyter
 - module, 101
- pybel.io.neo4j
 - module, 113
- pybel.io.nodelink
 - module, 93
- pybel.io.pykeen
 - module, 105
- pybel.io.pynpa
 - module, 102
- pybel.io.sbel
 - module, 94
- pybel.io.spia
 - module, 104
- pybel.io.triples
 - module, 106
- pybel.io.umbrella_nodelink
 - module, 114
- pybel.language
 - module, 146
- pybel.manager.database_io
 - module, 113
- pybel.manager.models
 - module, 123
- pybel.parser.modifiers
 - module, 162
- pybel.parser.modifiers.fragment
 - module, 30
- pybel.parser.modifiers.fusion
 - module, 31
- pybel.parser.modifiers.gene_modification
 - module, 27
- pybel.parser.modifiers.gene_substitution
 - module, 26
- pybel.parser.modifiers.location
 - module, 36
- pybel.parser.modifiers.protein_modification
 - module, 28
- pybel.parser.modifiers.protein_substitution
 - module, 27
- pybel.parser.modifiers.truncation
 - module, 29
- pybel.parser.modifiers.variant
 - module, 25
- pybel.struct
 - module, 11
- pybel.struct.filters
 - module, 43
- pybel.struct.getters
 - module, 70
- pybel.struct.grouping
 - module, 53
- pybel.struct.mutation.collapse
 - module, 73
- pybel.struct.mutation.deletion
 - module, 75
- pybel.struct.mutation.expansion
 - module, 77
- pybel.struct.mutation.induction
 - module, 79
- pybel.struct.mutation.induction_expansion
 - module, 83
- pybel.struct.mutation.inference
 - module, 85
- pybel.struct.mutation.metadata
 - module, 87
- pybel.struct.mutation.utils
 - module, 71
- pybel.struct.pipeline.decorators
 - module, 59
- pybel.struct.pipeline.exc
 - module, 60
- pybel.struct.query
 - module, 61
- pybel.struct.summary
 - module, 65
- PYBEL_EDGE_ALL_KEYS (*in module pybel.constants*), 142
- PYBEL_EDGE_DATA_KEYS (*in module pybel.constants*), 142
- PYBEL_EDGE_METADATA_KEYS (*in module pybel.constants*), 142
- PYBEL_NODE_DATA_KEYS (*in module pybel.constants*), 141
- PYBEL_NODE_FUNCTIONS (*in module pybel.constants*), 141
- PYBEL_VARIANT_KINDS (*in module pybel.constants*), 140
- pybel_version (*pybel.BELGraph property*), 16
- pybel-compile command line option
 - allow-naked-names, 132
 - disallow-nested, 132
 - disallow-unqualified-translocations, 132
 - no-citation-clearing, 132
 - no-identifier-validation, 132
 - required-annotations <required_annotations>, 132
 - skip-tqdm, 132
 - upgrade-urls, 132
 - verbose, 132
 - r, 132
 - v, 132
 - PATH, 132
- pybel-insert command line option
 - path, 132
- pybel-machine command line option
 - host <host>, 133
 - local, 133

AGENTS, 133

pybel-manage-drop command line option
--yes, 133

pybel-manage-edges-ls command line option
--limit <limit>, 134
--offset <offset>, 134

pybel-manage-examples command line option
--debug, 134
-v, 134

pybel-manage-namespaces-drop command line option
URL, 134

pybel-manage-namespaces-insert command line option
URL, 135

pybel-manage-namespaces-ls command line option
--namespace-id <namespace_id>, 135
--url <url>, 135
-i, 135
-u, 135

pybel-manage-networks-drop command line option
--network-id <network_id>, 135
--yes, 135
-n, 135
-y, 135

pybel-neo command line option
--connection <connection>, 136
--password <password>, 136
path, 136

pybel-parse command line option
--pprint, 137
TEXT, 137

pybel-serialize command line option
--bel <bel>, 137
--edgelist <edgelist>, 137
--graphml <graphml>, 137
--gsea <gsea>, 137
--nodelink <nodelink>, 137
--sif <sif>, 137
--tsv <tsv>, 137
path, 137

pybel-summarize command line option
path, 138

pybel-upload command line option
--host <host>, 138
--password <password>, 138
--user <user>, 138
path, 138

pybel-warnings command line option
path, 138

PyBELWarning, 175

Q

Query (class in *pybel.struct.query*), 61

query_citations() (*pybel.manager.QueryManager* method), 121

query_edges() (*pybel.manager.QueryManager* method), 121

query_edges_by_pubmed_identifiers() (*pybel.manager.QueryManager* method), 122

query_induction() (*pybel.manager.QueryManager* method), 122

query_neighbors() (*pybel.manager.QueryManager* method), 122

query_nodes() (*pybel.manager.QueryManager* method), 120

query_singleton_edges_from_network() (*pybel.manager.NetworkManager* method), 119

query_url (*pybel.manager.models.Namespace* attribute), 124

QueryManager (class in *pybel.manager*), 120

QueryMissingNetworksError, 61

R

raise_for_invalid_annotation_value() (*pybel.parser.parse_control.ControlParser* method), 158

raise_for_missing_citation() (*pybel.parser.parse_control.ControlParser* method), 159

raise_for_missing_name() (*pybel.parser.parse_concept.ConceptParser* method), 161

raise_for_missing_namespace() (*pybel.parser.parse_concept.ConceptParser* method), 161

raise_for_redefined_annotation() (*pybel.parser.parse_metadata.MetadataParser* method), 156

raise_for_redefined_namespace() (*pybel.parser.parse_metadata.MetadataParser* method), 156

raise_for_undefined_annotation() (*pybel.parser.parse_control.ControlParser* method), 158

raise_for_version() (*pybel.parser.parse_metadata.MetadataParser* method), 157

random() (*pybel.struct.graph.InduceDispatch* method), 23

RANGE_3P (in module *pybel.constants*), 140

RANGE_5P (in module *pybel.constants*), 140

rate_limit (*pybel.parser.parse_bel.BELParser* attribute), 153

RATE_LIMITING_STEP_OF (in module *pybel.constants*), 143

REACTANTS (in module *pybel.constants*), 140
 reactants (*pybel.parser.parse_bel.BELParser* attribute), 153
 Reaction (class in *pybel.dsl*), 173
 REACTION (in module *pybel.constants*), 141
 RedefinedAnnotationError, 176
 RedefinedNamespaceError, 175
 REGULATES (in module *pybel.constants*), 142
 RELATION (in module *pybel.constants*), 141
 relations() (*pybel.struct.graph.CountDispatch* method), 23
 remove_annotation_value() (in module *pybel.struct.mutation.metadata*), 87
 remove_associations() (in module *pybel.struct.mutation.deletion*), 75
 remove_biological_processes() (in module *pybel.struct.mutation.deletion*), 75
 remove_extra_citation_metadata() (in module *pybel.struct.mutation.metadata*), 87
 remove_filtered_edges() (in module *pybel.struct.mutation.deletion*), 75
 remove_filtered_nodes() (in module *pybel.struct.mutation.deletion*), 75
 remove_isolated_list_abundances() (in module *pybel.struct.mutation.deletion*), 75
 remove_isolated_nodes() (in module *pybel.struct.mutation.utils*), 71
 remove_isolated_nodes_op() (in module *pybel.struct.mutation.utils*), 71
 remove_non_causal_edges() (in module *pybel.struct.mutation.deletion*), 75
 remove_pathologies() (in module *pybel.struct.mutation.deletion*), 75
 REQUIRED_METADATA (in module *pybel.constants*), 145
 rev_abundance_labels (in module *pybel.constants*), 141
 Rna (class in *pybel.dsl*), 166
 RNA (in module *pybel.constants*), 141
 rna (*pybel.parser.parse_bel.BELParser* attribute), 152
 RnaFusion (class in *pybel.dsl*), 171
 run() (*pybel.Pipeline* method), 58
 run() (*pybel.struct.query.Query* method), 62
 run() (*pybel.struct.query.Seeding* method), 63

S

search_edges_with_bel() (*pybel.manager.QueryManager* method), 121
 search_edges_with_evidence() (*pybel.manager.QueryManager* method), 121
 Seeding (class in *pybel.struct.query*), 62
 SOURCE_MODIFIER (in module *pybel.constants*), 142
 source_modifier (*pybel.manager.models.Edge* attribute), 128

species (*pybel.manager.models.Namespace* attribute), 123
 statistics() (*pybel.struct.graph.SummarizeDispatch* method), 23
 store_bel() (*pybel.manager.models.Network* method), 125
 str() (*pybel.struct.graph.SummarizeDispatch* method), 24
 strip_annotations() (in module *pybel.struct.mutation.metadata*), 87
 SUBPROCESS_OF (in module *pybel.constants*), 143
 subprocess_of (*pybel.parser.parse_bel.BELParser* attribute), 153
 summarize (*pybel.BELGraph* property), 13
 summarize_node_filter() (in module *pybel.struct.filters*), 46
 SummarizeDispatch (class in *pybel.struct.graph*), 23
 summary() (*pybel.struct.graph.PlotDispatch* method), 24
 summary_dict() (*pybel.BELGraph* method), 22
 summary_str() (*pybel.BELGraph* method), 22
 survivors_are_inconsistent() (in module *pybel.struct.mutation.collapse*), 73

T

TARGET_MODIFIER (in module *pybel.constants*), 142
 target_modifier (*pybel.manager.models.Edge* attribute), 128

TEXT

pybel-parse command line option, 137
 text (*pybel.manager.models.Evidence* attribute), 127
 title (*pybel.manager.models.Citation* attribute), 126
 to_bel_commons() (in module *pybel*), 108
 to_bel_script() (in module *pybel*), 91
 to_biodati() (in module *pybel*), 109
 to_bytes() (in module *pybel*), 92
 to_bytes_gz() (in module *pybel*), 92
 to_csv() (in module *pybel*), 115
 to_cx() (in module *pybel*), 95
 to_cx_file() (in module *pybel*), 96
 to_cx_gz() (in module *pybel*), 96
 to_cx_jsons() (in module *pybel*), 95
 to_database() (in module *pybel*), 113
 to_edgelist() (in module *pybel*), 107
 to_graphdati() (in module *pybel*), 98
 to_graphdati_file() (in module *pybel*), 99
 to_graphdati_gz() (in module *pybel*), 99
 to_graphdati_jsonl() (in module *pybel*), 99
 to_graphdati_jsonl_gz() (in module *pybel*), 99
 to_graphdati_jsons() (in module *pybel*), 99
 to_graphml() (in module *pybel*), 114
 to_gsea() (in module *pybel*), 115
 to_hipathia() (in module *pybel*), 104
 to_hipathia_dfs() (in module *pybel*), 104
 to_indra_statements() (in module *pybel*), 101

- [to_indra_statements_json\(\)](#) (in module *pybel*), 101
[to_indra_statements_json_file\(\)](#) (in module *pybel*), 101
[to_jgif\(\)](#) (in module *pybel*), 96
[to_jgif_file\(\)](#) (in module *pybel*), 97
[to_jgif_gz\(\)](#) (in module *pybel*), 97
[to_jgif_jsons\(\)](#) (in module *pybel*), 97
[to_json\(\)](#) (*pybel.manager.models.Citation* method), 127
[to_json\(\)](#) (*pybel.manager.models.Edge* method), 128
[to_json\(\)](#) (*pybel.manager.models.Evidence* method), 127
[to_json\(\)](#) (*pybel.manager.models.Namespace* method), 124
[to_json\(\)](#) (*pybel.manager.models.NamespaceEntry* method), 124
[to_json\(\)](#) (*pybel.manager.models.Network* method), 125
[to_json\(\)](#) (*pybel.manager.models.Node* method), 126
[to_json\(\)](#) (*pybel.Pipeline* method), 58
[to_json\(\)](#) (*pybel.struct.query.Query* method), 62
[to_json\(\)](#) (*pybel.struct.query.Seeding* method), 63
[to_jupyter\(\)](#) (in module *pybel*), 101
[to_neo4j\(\)](#) (in module *pybel*), 113
[to_nodelink\(\)](#) (in module *pybel*), 93
[to_nodelink_file\(\)](#) (in module *pybel*), 93
[to_nodelink_gz\(\)](#) (in module *pybel*), 94
[to_nodelink_jsons\(\)](#) (in module *pybel*), 93
[to_npa_dfs\(\)](#) (in module *pybel*), 102
[to_npa_directory\(\)](#) (in module *pybel*), 102
[to_pickle\(\)](#) (in module *pybel*), 93
[to_pickle_gz\(\)](#) (in module *pybel*), 93
[to_s3\(\)](#) (in module *pybel*), 108
[to_sbel\(\)](#) (in module *pybel*), 94
[to_sbel_file\(\)](#) (in module *pybel*), 94
[to_sbel_gz\(\)](#) (in module *pybel*), 95
[to_sif\(\)](#) (in module *pybel*), 115
[to_spia_dfs\(\)](#) (in module *pybel*), 104
[to_spia_excel\(\)](#) (in module *pybel*), 105
[to_spia_tsvs\(\)](#) (in module *pybel*), 105
[to_triples\(\)](#) (in module *pybel*), 106
[to_triples_file\(\)](#) (in module *pybel*), 107
[to_umbrella_nodelink\(\)](#) (in module *pybel*), 114
[to_umbrella_nodelink_file\(\)](#) (in module *pybel*), 114
[to_umbrella_nodelink_gz\(\)](#) (in module *pybel*), 114
[Transcribable](#) (class in *pybel.dsl*), 166
[transcribed](#) (*pybel.parser.parse_bel.BELParser* attribute), 153
[TRANSCRIBED_TO](#) (in module *pybel.constants*), 142
[transformation\(\)](#) (in module *pybel.struct.pipeline.decorators*), 59
[transitivities](#) (*pybel.BELGraph* attribute), 13
[translated](#) (*pybel.parser.parse_bel.BELParser* attribute), 153
[TRANSLATED_TO](#) (in module *pybel.constants*), 142
[translocation](#) (*pybel.parser.parse_bel.BELParser* attribute), 152
[true_edge_predicate\(\)](#) (in module *pybel.struct.filters*), 44
[true_node_predicate\(\)](#) (in module *pybel.struct.filters*), 51
[TRUNCATION_POSITION](#) (in module *pybel.constants*), 145
[TWO_WAY_RELATIONS](#) (in module *pybel.constants*), 144
[type](#) (*pybel.manager.models.Node* attribute), 125
- ## U
- [uncachable_namespaces](#) (*pybel.parser.parse_metadata.MetadataParser* attribute), 155
[UndefinedAnnotationWarning](#), 177
[UndefinedNamespaceWarning](#), 176
[uni_in_place_transformation\(\)](#) (in module *pybel.struct.pipeline.decorators*), 59
[uni_transformation\(\)](#) (in module *pybel.struct.pipeline.decorators*), 59
[union\(\)](#) (in module *pybel.struct*), 55
[union\(\)](#) (*pybel.Pipeline* static method), 59
[UNQUALIFIED_EDGES](#) (in module *pybel.constants*), 144
[uploaded](#) (*pybel.manager.models.Namespace* attribute), 123
- ## URL
- [pybel-manage-namespaces-drop](#) command line option, 134
[pybel-manage-namespaces-insert](#) command line option, 135
[url](#) (*pybel.manager.models.Namespace* attribute), 123
- ## V
- [validate_unset_command\(\)](#) (*pybel.parser.parse_control.ControlParser* method), 159
[Variant](#) (class in *pybel.dsl*), 168
[VARIANTS](#) (in module *pybel.constants*), 140
[variants\(\)](#) (*pybel.struct.graph.CountDispatch* method), 23
[version](#) (*pybel.BELGraph* property), 14
[version](#) (*pybel.manager.models.Namespace* attribute), 123
[version](#) (*pybel.manager.models.Network* attribute), 125
[VersionFormatWarning](#), 178
[volume](#) (*pybel.manager.models.Citation* attribute), 126
- ## W
- [warnings](#) (*pybel.BELGraph* property), 16

X

XREFS (*in module `pybel.constants`*), [140](#)