
PyBEL Documentation

Release 0.7.1-dev

Charles Tapley Hoyt

Jul 20, 2017

1	Links	3
1.1	Installation	3
1.2	Overview	4
1.3	Data Model	9
1.4	Input and Output	22
1.5	Cookbook	31
1.6	Troubleshooting	32
1.7	Logging Messages	34
1.8	Constants	39
1.9	Parsers	46
1.10	Cache	53
1.11	Utilities	59
1.12	Roadmap	63
1.13	Technology	64
2	Indices and Tables	67
	Python Module Index	69

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 [IMI](#) project, [AETIONOMY](#), to describe complex disease networks with several thousands of relationships.

PyBEL is a Python software package that parses BEL scripts, 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.

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

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

Installation

PyBEL is tested on both Python3 and legacy Python2 installations on Mac OS and Linux using [Travis CI](#).

Warning: PyBEL is not thoroughly tested on Windows.

Installation

Easiest

Download the latest stable code from [PyPI](#) with:

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

Get the Latest

Download the most recent code from [GitHub](#) with:

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

For Developers

Clone the repository from [GitHub](#) and install in editable mode with:

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

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.

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 either by

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

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

Overview

Background on Systems Biology Modelling

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 in the [IMI](#) project, [AETIONOMY](#), to describe complex disease networks with several thousands of relationships. For a detailed explanation, see the [BEL 1.0](#) and [2.0](#) specifications.

OpenBEL Links

- [OpenBEL on Google Groups](#)
- [OpenBEL Wiki](#)
- [OpenBEL on GitHub](#)
- [Chat on Gitter](#)

Design Considerations

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.

Outdated Namespaces

OpenBEL provides a variety of namespaces covering each of the BEL function types. These namespaces are generated by code found at <https://github.com/OpenBEL/resource-generator> and distributed at <http://resources.openbel.org/belframework/>.

This code has not been maintained to reflect the changes in the underlying resources, so this repository has been forked and updated at <https://github.com/pybel/resource-generator> to reflect the most recent versions of the underlying namespaces. The files are now distributed using the Fraunhofer SCAI ownCloud server. An example list of these namespaces can be found in the [template BEL script](#) in the [PyBEL Resources](#) repository on GitHub.

Generating New Namespaces

In some cases, it is appropriate to design a new namespace, using the [custom namespace specification](#) provided by the OpenBEL Framework. Scripts for generating additional resource files have been posted to the [PyBEL Notebooks](#) repository on GitHub.

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>)

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.

Extensions to BEL

The PyBEL compiler is fully compliant with both BEL v1.0 and v2.0 and automatically upgrades legacy statements. Additionally, we have included 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.

Syntax for Epigenetics

We introduce 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/>

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. We introduce 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]+"
```

Definition of Resources using OWL

One constraint imposed by the BEL language is that definitions of namespaces and annotations must follow a specific format. However, the creation and maintenance of terminologies in the biological domain has tended towards the usage of the Web Ontology Format (OWL). Services such as the Ontology Lookup Service allow for standardized querying and search of these resources, and provide an important semantic integration layer that previous software tools for BEL did not include. PyBEL allows for these resources to be named directly in definitions with the following syntax:

```
DEFINE ANNOTATION CELL AS OWL "http://purl.obolibrary.org/obo/cl/releases/2016-11-23/cl.owl"
DEFINE NAMESPACE DO AS OWL "http://purl.obolibrary.org/obo/doid/releases/2017-05-05/doid.owl"
```

This allows PyBEL to import the semantic information from the ontology as well, and provide much more rich algorithms that take into account the hierarchy and synonyms provided.

PyBEL uses the `onto2nx` package to parse OWL documents in many different formats, including OWL/XML, RDF/XML, and RDF.

Explicit Node Labels

While the BEL 2.0 specification made it possible to represent new terms, such as the APOE gene with two variants resulting in the E2 allele, it came at the price of encoding terms in a technical and less readable way. An explicit statement for labeling nodes has been added, such that the resulting data structure will have a label for the node:

```
g(HGNC:APOE, var(c.388T>C), var(c.526C>T)) labeled "APOE E2"
```

When InChI is used, these strings are very hard to visualize. Using a label is helpful for later visualization:

```
a(INCHI:"InChI=1S/C20H28N2O5/c1-3-27-20(26)16(12-11-15-8-5-4-6-9-15)21-14(2)18(23)22-13-7-14-6,8-9,14,16-17,21H,3,7,10-13H2,1-2H3,(H,24,25)/t14-,16-,17-/m0/s1") labeled "Enalapril"
```

Below is the same molecule again, but represented with an InChIKey:

```
a(INCHIKEY:"GBXSMTUPTTWBMN-XIRDDKMYSAN") labeled "Enalapril"
```

It's also easy to use the universe of RESTful API services from UniChem, ChEMBL, or WikiData to download and annotate these automatically. For further information on Enalapril can be found [WikiData](#), [UniChem](#), and [ChEMBL](#).

Things to Consider

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.

Multiple Annotations

When an annotation has a list, it means that the following BEL relations are true for each of the listed values. The lines below show a BEL relation that corresponds to two edges, each with the same citation but different values for ExampleAnnotation. This should be considered carefully for analyses dealing with the number of edges between two entities.

```
SET Citation = {"PubMed", "Example Article", "12345"}
SET ExampleAnnotation = {"Example Value 1", "Example Value 2"}
p(HGNC:YFG1) -> p(HGNC:YFG2)
```

Furthermore, if there are multiple annotations with lists, the following BEL relations are true for all of the different combinations of them. The following statements will produce four edges, as the cartesian product of the values used for both ExampleAnnotation1 and ExampleAnnotation2. This might not be the knowledge that the annotator wants to express, and is prone to mistakes, so use of annotation lists are not recommended.

```
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)
```

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”.

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 an entire 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, we were using directly increasing 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

We considered the ambiguity of nested statements to be too great of a risk to include their usage in the PyBEL compiler. In our group 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.

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 Knowledge Base does an exemplary job of this. Ultimately, using non-standard URL's 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.

Data Model

Molecular biology is a directed graph; not a table. BEL expresses how biological entities interact within many different contexts, with descriptive annotations. PyBEL represents data as a directional multigraph using an extension of `networkx.MultiDiGraph`. Each node and edge has an associated data dictionary for storing relevant/contextual information.

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.

Constants

These documents refer to many aspects of the data model using constants, which can be found in the top-level module `pybel.constants`. In these examples, all constants are imported with the following code:

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

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.

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
a	ABUNDANCE
abundance	ABUNDANCE
geneAbundance	GENE
g	GENE
rnaAbundance	RNA
r	RNA
microRNAAbundance	MIRNA
m	MIRNA
proteinAbundance	PROTEIN
p	PROTEIN
biologicalProcess	BIOPROCESS
bp	BIOPROCESS
pathology	PATHOLOGY
path	PATHOLOGY
complexAbundance	COMPLEX
complex	COMPLEX
compositeAbundance	COMPOSITE
composite	COMPOSITE
reaction	REACTION
rxn	REACTION

Graph

PyBEL's main data structure is a subclass of `networkx.MultiDiGraph`.

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.

class `pybel.BELGraph` (*data=None, **kwargs*)

This class represents biological knowledge assembled in BEL as a network by extending the `networkx.MultiDiGraph`.

The default constructor parses a BEL graph using the built-in `networkx` methods. For IO, see the `pybel.io` module

Parameters

- **data** – initial graph data to pass to `networkx.MultiDiGraph`
- **kwargs** – keyword arguments to pass to `networkx.MultiDiGraph`

has_singleton_terms = None

Is true if during BEL Parsing, a term that is not part of a relation is found

document

A dictionary holding the metadata from the “Document” section of the BEL script. All keys are normalized according to `pybel.constants.DOCUMENT_KEYS`

name

Gets the graph’s name. Requires a weird hack in the signature since it’s overriding a property

version

The graph’s version, from the SET DOCUMENT Version = "...” entry in the source BEL script

description

The graph’s description, from the SET DOCUMENT Description = "...” entry in the source BEL Script

namespace_url

A dictionary mapping the keywords used to create this graph to the URLs of the BELNS file

namespace_owl

A dictionary mapping the keywords used to create this graph to the URLs of the OWL file

namespace_pattern

A dictionary mapping the namespace keywords used to create this graph to their regex patterns

annotation_url

A dictionary mapping the annotation keywords used to create this graph to the URLs of the BELANNO file

annotation_owl

A dictionary mapping the annotation keywords to the URL of the OWL file

annotation_pattern

A dictionary mapping the annotation keywords used to create this graph to their regex patterns

annotation_list

A dictionary mapping the keywords of locally defined annotations to a set of their values

pybel_version

Stores the version of PyBEL with which this graph was produced as a string

warnings

Warnings are stored in a list of 4-tuples that is a property of the graph object. This tuple respectively contains the line number, the line text, the exception instance, and the context dictionary from the parser at the time of error.

add_unqualified_edge (*u, v, relation*)

Adds unique edge that has no annotations

Parameters

- **u** (*tuple*) – The source BEL node
- **v** (*tuple*) – The target BEL node
- **relation** (*str*) – A relationship label from `pybel.constants`

add_simple_node (*function, namespace, name*)
Adds a simple node, with only a namespace and name

Parameters

- **function** (*str*) – The node’s function (*pybel.constants.GENE*, *pybel.constants.RNA*, *pybel.constants.PROTEIN*, etc)
- **namespace** (*str*) – The namespace
- **name** (*str*) – The name of the node

has_edge_citation (*u, v, key*)
Does the given edge have a citation?

get_edge_citation (*u, v, key*)
Gets the citation for a given edge

has_edge_evidence (*u, v, key*)
Does the given edge have evidence?

get_edge_evidence (*u, v, key*)
Gets the evidence for a given edge

get_edge_annotations (*u, v, key*)
Gets the annotations for a given edge

get_node_name (*node*)
Gets the node’s name, or return None if no name

get_node_label (*node*)
Gets the label for a given node

set_node_label (*node, label*)
Sets the label for a given node

get_node_description (*node*)
Gets the description for a given node

set_node_description (*node, description*)
Sets the description for a given node

Nodes

Nodes are used to represent physical entities’ abundances. The relevant data about a node is stored in its associated dictionary in *networkx*. After parsing, `p (HGNC:GSK3B)` becomes:

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

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 PTM, gene modification, fragment, or HGVS variant. The ‘kind’ value for a variant is ‘hgvs’, but best described by *pybel.constants.HGVS*

For example, the node `p (HGNC:GSK3B, var (p.Gly123Arg))` is represented with the following:

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

See also:

- [BEL 2.0 specification on variants](#)
- [HGVS conventions](#)

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:

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

See also:

[BEL 2.0 specification on gene 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:

```
{
  FUNCTION: GENE,
  NAMESPACE: 'HGNC',
  NAME: 'APP',
  VARIANTS: [
```

```
{
  KIND: HGVS,
  IDENTIFIER: 'p.Arg275His'
}
]
```

See also:

BEL 2.0 specification on [protein substitutions](#)

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:

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

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](#)

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:

```
{
  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:

```

{
    FUNCTION: PROTEIN,
    NAMESPACE: 'HGNC',
    NAME: 'GSK3B',
    VARIANTS: [
        {
            KIND: FRAGMENT,
            FRAGMENT_MISSING: '?',
        }
    ]
}

```

See also:

BEL 2.0 specification on [proteolytic fragments \(2.2.3\)](#)

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:

```

{
    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.

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:

```
{
  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](#)

Fusions

Gene, RNA, protein, and miRNA 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:

```

{
    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 \(2.6.1\)](#)

Unqualified Edges

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

Variant and Modifications' Parent Relations

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

List Abundances

Complexes and composites that are defined by lists do not receive information about the identifier, and are only described by their function. `complex(p(HGNC:FOS), p(HGNC:JUN))` becomes:

```
{
  FUNCTION: COMPLEX
}
```

The following edges are 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 specification on [complex abundances](#)

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

```
{
  FUNCTION: COMPOSITE
}
```

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)
```

See also:

BEL 2.0 specification on [composite abundances](#)

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(+)")))
```

results in the following data:

```
{
  FUNCTION: REACTION
}
```

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 specification on [reactions](#)

Edges

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), GOCC:intracellular, GOCC:"cell membrane")` also induced `tloc(p(HGNC:YFG), GOCC:intracellular, GOCC:"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), GOCC:intracellular, GOCC:"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`

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:

```
{
  SUBJECT: {
    # ... modifications to the subject node. Only present if non-empty.
  },
  RELATION: 'positiveCorrelation',
  OBJECT: {
    # ... modifications to the object node. Only present if non-empty.
  },
  EVIDENCE: '...',
  CITATION : {
    # ... citation information
  }
  ANNOTATIONS: {
    # ... additional annotations as key:value pairs
  }
}
```

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:

```
{
  RELATION: 'positiveCorrelation',
  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:

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

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(GOCC:lysozome)) pos act(p(HGNC:GSK3B), ma(kin))` becomes:

```
{
  SUBJECT: {
    LOCATION: {
      NAMESPACE: 'GOCC',
      NAME: 'lysozome'
    }
  },
  RELATION: 'positiveCorrelation',
  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(GOCC:intracellular), toLoc(GOCC:"cell membrane")) -> p(HGNC:B)`
- `p(HGNC:A, location(GOCC:"cell membrane")) -> p(HGNC:B)`

See also:

BEL 2.0 specification on [cellular location \(2.2.4\)](#)

Translocations

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

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

See also:

BEL 2.0 specification on [translocations](#)

Degradations

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

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

See also:

BEL 2.0 specification on [degradations](#)

Input and Output

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.

Import

```
pybel.from_lines (lines, manager=None, allow_naked_names=False, allow_nested=False,
                 allow_unqualified_translocations=False, citation_clearing=True,
                 warn_on_singleton=True, no_identifier_validation=False, **kwargs)
```

Loads a BEL graph from an iterable over the lines of a BEL script

Parameters

- **lines** (*iter[str]*) – An iterable of strings (the lines in a BEL script)
- **manager** (str or `pybel.manager.CacheManager` or `pybel.parser.MetadataParser`) – database connection string to cache, pre-built `CacheManager`, pre-built `MetadataParser` or `None` to use default cache
- **allow_naked_names** (*bool*) – if true, turn off naked namespace failures
- **allow_nested** (*bool*) – if true, turn off nested statement failures
- **allow_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`
- **warn_on_singleton** (*bool*) – Should the parser thorough warnings on singletons? Only disable this if you're sure your BEL Script is syntactically and semantically valid.
- **kwargs** (*dict*) – keyword arguments to pass to `networkx.MultiDiGraph`

Returns A BEL graph

Return type *BELGraph*

```
pybel.from_path (path, manager=None, allow_naked_names=False, allow_nested=False,
                citation_clearing=True, warn_on_singleton=True, no_identifier_validation=False,
                encoding='utf-8', **kwargs)
```

Loads a BEL graph from a file resource. This function is a thin wrapper around `from_lines()`.

Parameters

- **path** (*str*) – A file path
- **manager** (str or `pybel.manager.CacheManager` or `pybel.parser.MetadataParser`) – database connection string to cache, pre-built `CacheManager`, pre-built `MetadataParser` or `None` to use default cache
- **allow_naked_names** (*bool*) – if true, turn off naked namespace failures
- **allow_nested** (*bool*) – if true, turn off nested statement failures

- **citation_clearing** (*bool*) – Should SET Citation statements clear evidence and all annotations? Delegated to `pybel.parser.ControlParser`
- **warn_on_singleton** (*bool*) – Should the parser thorough warnings on singletons? Only disable this if you're sure your BEL Script is syntactically and semantically valid.
- **encoding** (*str*) – the encoding to use when reading this file. Is passed to `codecs.open`. See the python docs for a list of standard encodings. For example, files starting with a UTF-8 BOM should use `utf_8_sig`
- **kwargs** (*dict*) – Keyword arguments to pass to `networkx.MultiDiGraph`

Returns A BEL graph

Return type *BELGraph*

`pybel.from_url(url, manager=None, allow_naked_names=False, allow_nested=False, citation_clearing=True, warn_on_singleton=True, **kwargs)`

Loads a BEL graph from a URL resource. This function is a thin wrapper around `from_lines()`.

Parameters

- **url** (*str*) – A valid URL pointing to a BEL resource
- **manager** (*str* or `pybel.manager.CacheManager` or `pybel.parser.MetadataParser`) – database connection string to cache, pre-built `CacheManager`, pre-built `MetadataParser` or `None` to use default cache
- **allow_naked_names** (*bool*) – if true, turn off naked namespace failures
- **allow_nested** (*bool*) – if true, turn off nested statement failures
- **citation_clearing** (*bool*) – Should SET Citation statements clear evidence and all annotations? Delegated to `pybel.parser.ControlParser`
- **warn_on_singleton** (*bool*) – Should the parser thorough warnings on singletons? Only disable this if you're sure your BEL Script is syntactically and semantically valid.
- **kwargs** (*dict*) – Keyword arguments to pass to `networkx.MultiDiGraph`

Returns A BEL graph

Return type *BELGraph*

Canonicalization

`pybel.to_bel_lines(graph)`

Returns an iterable over the lines of the BEL graph as a canonical BEL Script (.bel)

Parameters **graph** (*BELGraph*) – the BEL Graph to output as a BEL Script

Returns An iterable over the lines of the representative BEL script

Return type `iter[str]`

`pybel.to_bel(graph, file=None)`

Outputs the BEL graph as canonical BEL to the given file/file-like/stream. Defaults to standard out.

Parameters

- **graph** (*BELGraph*) – the BEL Graph to output as a BEL Script
- **file** (*file*) – A writable file-like object. If `None`, defaults to standard out.

`pybel.to_bel_path` (*graph*, *path*)

Writes the BEL graph as a canonical BEL Script to the given path

Parameters

- **graph** (*BELGraph*) – the BEL Graph to output as a BEL Script
- **path** (*str*) – A file path

Transport

All transport pairs are reflective and data-preserving.

Bytes

This module contains IO functions for interconversion between BEL graphs and python pickle objects

`pybel.from_pickle` (*path*, *check_version=True*)

Reads a graph from a gpickle file.

Parameters

- **or str path** (*file*) – 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

Returns A BEL graph

Return type *BELGraph*

`pybel.to_pickle` (*graph*, *file*, *protocol=4*)

Writes this graph to a pickle object with `networkx.write_gpickle()`. 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
- **file** (*file or file-like or str*) – A file or filename to write to
- **protocol** (*int*) – Pickling protocol to use

See also:

<https://docs.python.org/3.6/library/pickle.html#data-stream-format>

`pybel.from_bytes` (*bytes_graph*, *check_version=True*)

Reads 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

Returns A BEL graph

Return type *BELGraph*

`pybel.to_bytes` (*graph*, *protocol=4*)

Converts 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 network
- **protocol** (*int*) – Pickling protocol to use

Returns Pickled bytes representing the graph

Return type *bytes*

See also:

<https://docs.python.org/3.6/library/pickle.html#data-stream-format>

Node-Link JSON

This module contains IO functions for interconversion between BEL graphs and Node-Link JSON

`pybel.from_json` (*graph_json_dict*, *check_version=True*)

Builds a graph from Node-Link JSON Object

Parameters

- **graph_json_dict** (*dict*) – A JSON dictionary representing a graph
- **check_version** (*bool*) – Checks if the graph was produced by this version of PyBEL

Returns A BEL graph

Return type *BELGraph*

`pybel.to_json` (*graph*)

Converts this graph to a Node-Link JSON object

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

Returns A Node-Link JSON object representing the given graph

Return type *dict*

`pybel.from_json_file` (*file*, *check_version=True*)

Builds a graph from the Node-Link JSON contained in the given file

Parameters

- **file** (*file*) – A readable file or file-like
- **check_version** (*bool*) – Checks if the graph was produced by this version of PyBEL

Returns A BEL graph

Return type *BELGraph*

`pybel.to_json_file` (*graph*, *file*)

Writes this graph as Node-Link JSON to a file

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **file** (*file*) – A write-supporting file or file-like object

`pybel.from_jsons` (*graph_json_str*, *check_version=True*)

Reads a BEL graph from a Node-Link JSON string

Parameters

- **graph_json_str** (*str*) – A Node-Link JSON string produced by PyBEL

- **check_version** (*bool*) – Checks if the graph was produced by this version of PyBEL

Returns A BEL graph

Return type *BELGraph*

`pybel.to_jsons` (*graph*)

Dumps this graph as a Node-Link JSON object to a string

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

Returns A string representation of the Node-Link JSON produced for this graph by `pybel.to_json()`

Return type *str*

JSON Graph Interchange Format

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*)

Builds a BEL graph from a JGIF JSON object.

Parameters **graph_jgif_dict** (*dict*) – The JSON object representing the graph in JGIF format

Returns A BEL graph

Return type *BELGraph*

`pybel.from_cbn_jgif` (*graph_jgif_dict*)

Maps 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

Returns A BEL graph

Return type *BELGraph*

Example:

```
>>> 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)
```

`pybel.to_jgif` (*graph*)

Builds a JGIF dictionary from a BEL graph.

Parameters **graph** (*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.

Example:

```
>>> import pybel, os, json
>>> graph_url = 'https://arty.scai.fraunhofer.de/artifactory/bel/knowledge/
↳selventa-small-corpus/selventa-small-corpus-20150611.bel'
>>> graph = pybel.from_url(graph_url)
>>> graph_jgif_json = pybel.to_jgif(graph)
>>> with open(os.path.expanduser('~/Desktop/small_corpus.json'), 'w') as f:
...     json.dump(graph_jgif_json, f)
```

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)`

Rebuilds a BELGraph from CX JSON output from PyBEL

Parameters `cx` (*list*) – The CX JSON for this graph

Returns A BEL graph

Return type *BELGraph*

`pybel.to_cx(graph)`

Converts BEL Graph to CX data format (as in-memory JSON) for use with NDEx

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

Returns The CX JSON for this graph

Return type *list*

See also:

- [NDEx Python Client](#)
- [PyBEL / NDEx Python Client Wrapper](#)

`pybel.from_cx_file(file)`

Reads a file containing CX JSON and converts to a BEL graph

Parameters `file` (*file*) – A readable file or file-like containing the CX JSON for this graph

Returns A BEL Graph representing the CX graph contained in the file

Return type *BELGraph*

`pybel.to_cx_file(graph, file)`

Writes this graph to a JSON file in CX format

Parameters

- `graph` (*BELGraph*) – A BEL graph

- **file** (*file*) – A writable file or file-like

Example:

```
>>> from pybel import from_url, to_cx_file
>>> from pybel.constants import SMALL_CORPUS_URL
>>> graph = from_url(SMALL_CORPUS_URL)
>>> with open('graph.cx', 'w') as f:
>>> ... to_cx_file(graph, f)
```

`pybel.from_cx_jsons` (*graph_cx_json_str*)

Reconstitutes a BEL graph from a CX JSON string

Parameters `graph_cx_json_str` (*str*) – CX JSON string

Returns A BEL graph representing the CX graph contained in the string

Return type *BELGraph*

`pybel.to_cx_jsons` (*graph*)

Dumps a BEL graph as CX JSON to a string

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

Returns CX JSON string

Return type *str*

Export

`pybel.to_graphml` (*graph, file*)

Writes this graph to GraphML XML file using `networkx.write_graphml()`. The `.graphml` file extension is suggested so Cytoscape can recognize it.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **file** (*file*) – A file or file-like object

`pybel.to_csv` (*graph, file*)

Writes the graph as a tab-separated edge list with the 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.

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **file** (*file*) – A writable file or file-like.

`pybel.to_sif` (*graph, file*)

Writes the graph as a tab-separated SIF file with 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.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **file** (`file`) – A writable file or file-like.

`pybel.to_gsea` (`graph`, `file`)

Writes the genes/gene products to a GRP file for use with GSEA gene set enrichment analysis

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **file** (`file`) – A writeable file or file-like object

See also:

- [GRP format specification](#)
- [GSEA publication](#)

Database

SQL Database

This module provides IO functions to the relational edge store.

`pybel.from_database` (`name`, `version`, `connection=None`)

Loads a BEL graph from a database.

Parameters

- **name** (`str`) – The name of the graph
- **version** (`str`) – The version string of the graph. If not specified, loads most recent graph added with this name
- **connection** (`None` or `str` or `pybel.manager.cache.CacheManager`) – An RFC-1738 database connection string, a pre-built `CacheManager`, or `None` for default connection

Returns A BEL graph loaded from the database

Return type `BELGraph`

`pybel.to_database` (`graph`, `connection=None`, `store_parts=False`)

Stores a graph in a database.

Parameters

- **graph** (`BELGraph`) – A BEL graph
- **connection** (`None` or `str` or `pybel.manager.cache.CacheManager`) – An RFC-1738 database connection string, a pre-built `CacheManager`, or `None` for default connection
- **store_parts** (`bool`) – Should the graph be stored in the edge store?

Neo4j

This module contains IO functions for outputting BEL graphs to a Neo4J graph database

`pybel.to_neo4j` (*graph*, *neo_connection*, *context=None*)
 Uploads a BEL graph to Neo4J graph database using `py2neo`

Parameters

- **graph** (`BELGraph`) – A BEL Graph
- **neo_connection** (`py2neo.Graph`) – A `py2neo` connection object. Refer to the [py2neo documentation](#) for how to build this object.
- **context** (*str*) – A disease context to allow for multiple disease models in one neo4j instance. Each edge will be assigned an attribute `pybel_context` with this value

Example Usage:

```
>>> import pybel, py2neo
>>> url = 'http://resource.belframework.org/belframework/1.0/knowledge/small_
↳corpus.bel'
>>> g = pybel.from_url(url)
>>> neo_graph = py2neo.Graph("http://localhost:7474/db/data/") # use your own_
↳connection settings
>>> pybel.to_neo4j(g, neo_graph)
```

NDEx

This package provides a wrapper around `pybel.to_cx()` and NDEx [client](#) to allow for easy upload and download of BEL documents to the NDEx database.

The programmatic API also provides options for specifying username and password. By default, it checks the environment variables: `NDEX_USERNAME` and `NDEX_PASSWORD`.

`pybel.from_ndex` (*network_id*, *username=None*, *password=None*, *debug=False*)
 Downloads a BEL Graph from NDEx

Warning: This function only will work for CX documents that have been originally exported from PyBEL

Parameters

- **network_id** (*str*) – The UUID assigned to the network by NDEx
- **username** (*str*) – NDEx username
- **password** (*str*) – NDEx password
- **debug** (*bool*) – If true, turn on NDEx client debugging

Returns A BEL graph

Return type `BELGraph`

Example Usage:

```
>>> from pybel import from_ndex
>>> network_id = '1709e6f3-04a1-11e7-aba2-0ac135e8bacf'
>>> graph = from_ndex(network_id)
```

`pybel.to_ndex` (*graph*, *username=None*, *password=None*, *debug=False*)
 Uploads a BEL graph to NDEx

Parameters

- **graph** (*BELGraph*) – A BEL graph
- **username** (*str*) – NDEx username
- **password** (*str*) – NDEx password
- **debug** (*bool*) – If true, turn on NDEx client debugging

Returns The UUID assigned to the network by NDEx

Return type *str*

Example Usage:

```
>>> import pybel
>>> graph = pybel.from_url('http://resources.openbel.org/belframework/20150611/
↳knowledge/small_corpus.bel')
>>> pybel.to_ndex(graph)
```

Cookbook

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](#)

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"
```

Command Line

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/`.

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`.

Get the latest PyBEL Suite

Here is a list of all of the things you can install:

```
python3 -m pip uninstall pybel_web_validator
python3 -m pip uninstall pybel_tools
python3 -m pip uninstall pybel
python3 -m pip install git+https://github.com/pybel/pybel.git@develop
python3 -m pip install git+https://github.com/pybel/pybel-tools.git@develop
python3 -m pip install git+https://github.com/pybel/pybel-web-validator.git
```

Troubleshooting

Common problems and questions will be posted here.

Encoding Issues

Sometimes, Windows computers stick a weird unicode object `\u2013` at the beginning of files. This makes the function `pybel.parser.utils.sanitize_file_lines()` have a problem. The solution, when loading a BEL script via `pybel.from_path()` is to use the `encodings` keyword argument to specify the right encoding. The default is `utf-8` because this is the most common, but when this error happens, set it explicitly to `utf_8_sig`. More specific documentation is available in the [Inputs and Outputs](#) page.

Scenario

```
>>> import pybel
>>> graph = pybel.from_path('~\Desktop\small_corpus.bel')

UnicodeDecodeError Traceback (most recent call last) <ipython-input-11-99f2a76596b1>
-> in <module>()
      7 ad = pybel.from_pickle(path_2_AD_pickle)
      8 else:
---->  9 ad = pybel.from_path(path_2_AD_bel)
     10 pybel.to_pickle(ad, path_2_AD_pickle)

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\site-packages\pybel\graph.
-> py in from_path(path, **kwargs)
     61 log.info('Loading from path: %s', path)
     62 with open(os.path.expanduser(path)) as f:
-->  63 return BELGraph(lines=f, **kwargs)
     64
     65

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\site-packages\pybel\graph.
-> py in __init__(self, lines, context, lenient, definition_cache_manager, log_stream,
-> *attrs, **kwargs)
    102
    103 if lines is not None:
--> 104 self.parse_lines(lines, context, lenient,
--> definition_cache_manager, log_stream)
    105
    106 def parse_lines(self, lines, context=None, lenient=False, definition_cache_
-> manager=None, log_stream=None):
```

```

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\site-packages\pybel\graph.
↳py in parse_lines(self, lines, context, lenient, definition_cache_manager, log_
↳stream)
125 self.context = context
126
--> 127 docs, defs, states =
--> split_file_to_annotations_and_definitions(lines)
128
129 if isinstance(definition_cache_manager, DefinitionCacheManager):

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\site-
↳packages\pybel\parser\utils.py in split_file_to_annotations_and_definitions(file)
49 def split_file_to_annotations_and_definitions(file):
50 """Enumerates a line iterable and splits into 3 parts"""
--> 51 content = list(sanitize_file_lines(file))
52
53 end_document_section = 1 + max(j for j, (i, l) in enumerate(content) if l.
↳startswith('SET DOCUMENT'))

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\site-
↳packages\pybel\parser\utils.py in sanitize_file_lines(f)
16 it = iter(it)
17
--> 18 for line_number, line in it:
19 if line.endswith('\n'):
20 log.log(4, 'Multiline quote starting on line: %d', line_number)

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\site-
↳packages\pybel\parser\utils.py in <genexpr>(.0)
12 def sanitize_file_lines(f):
13 """Enumerates a line iterator and returns the pairs of (line number, line) that
↳are cleaned"""
--> 14 it = (line.strip() for line in f)
15 it = filter(lambda i_l: i_l[1] and not i_l[1].startswith('#'), enumerate(it,
↳start=1))
16 it = iter(it)

C:\Users\s8310253\AppData\Local\Continuum\Anaconda3420\lib\encodings\cp1252.py in
↳decode(self, input, final)
21 class IncrementalDecoder(codecs.IncrementalDecoder):
22 def decode(self, input, final=False):
--> 23 return
--> codecs.charmap_decode(input, self.errors, decoding_table)[0]
24
25 class StreamWriter(Codec, codecs.StreamWriter):

UnicodeDecodeError: 'charmap' codec can't decode byte 0x9d in position 4668:
↳character maps to <undefined>

```

Solution

```

>>> import pybel
>>> graph = pybel.from_path('~\Desktop\small_corpus.bel', encoding='utf_8_sig')
>>> # Success!

```

Logging Messages

Errors

This module contains base exceptions that are shared through the package

exception `pybel.exceptions.PyBelWarning`

The base class for warnings during compilation from which PyBEL can recover

Parse Exceptions

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.parser.parse_exceptions.PyBelParserWarning` (*line_number*, *line*, *position*, **args*)

Base PyBEL parser exception, which holds the line and position where a parsing problem occurred

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_exceptions.InconsistentDefinitionError` (*line_number*, *line*, *position*, *definition*)

Base PyBEL error for redefinition

exception `pybel.parser.parse_exceptions.RedefinedNamespaceError` (*line_number*, *line*, *position*, *definition*)

Raised when a namespace is redefined

exception `pybel.parser.parse_exceptions.RedefinedAnnotationError` (*line_number*, *line*, *position*, *definition*)

Raised when an annotation is redefined

exception `pybel.parser.parse_exceptions.NameWarning` (*line_number*, *line*, *position*, *name*, **args*)

The base class for errors related to nomenclature

exception `pybel.parser.parse_exceptions.NakedNameWarning` (*line_number*, *line*, *position*, *name*, **args*)

Raised when there is an identifier without a namespace. Enable lenient mode to suppress

exception `pybel.parser.parse_exceptions.MissingDefaultNameWarning` (*line_number*, *line*, *position*, *name*, **args*)

Raised if reference to value not in default namespace

exception `pybel.parser.parse_exceptions.NamespaceIdentifierWarning` (*line_number*,
line, *position*,
namespace,
name)

The base class for warnings related to namespace:name identifiers

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.parser.parse_exceptions.UndefinedNamespaceWarning` (*line_number*,
line, *position*,
namespace,
name)

Raised if reference made to undefined namespace

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.parser.parse_exceptions.MissingNamespaceNameWarning` (*line_number*,
line, *po-*
sition,
namespace,
name)

Raised if reference to value not in namespace

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.parser.parse_exceptions.MissingNamespaceRegexWarning` (*line_number*,
line, *po-*
sition,
namespace,
name)

Raised if reference not matching regex

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.parser.parse_exceptions.AnnotationWarning` (*line_number*, *line*, *position*, *annotation*, *args)

Base exception for annotation warnings

exception `pybel.parser.parse_exceptions.UndefinedAnnotationWarning` (*line_number*, *line*, *position*, *annotation*, *args)

Raised when an undefined annotation is used

exception `pybel.parser.parse_exceptions.MissingAnnotationKeyWarning` (*line_number*, *line*, *position*, *annotation*, *args)

Raised when trying to unset an annotation that is not set

exception `pybel.parser.parse_exceptions.AnnotationIdentifierWarning` (*line_number*, *line*, *position*, *annotation*, *value*)

Base exception for annotation:value pairs

exception `pybel.parser.parse_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.

exception `pybel.parser.parse_exceptions.MissingAnnotationRegexWarning` (*line_number*, *line*, *position*, *annotation*, *value*)

Raised if annotation doesn’t match regex

exception `pybel.parser.parse_exceptions.VersionFormatWarning` (*line_number*, *line*, *position*, *version_string*)

Raised if the version string doesn’t adhere to semantic versioning or YYYYMMDD format

exception `pybel.parser.parse_exceptions.MalformedMetadataException` (*line_number*, *line*, *args)

Raised when an invalid metadata line is encountered

exception `pybel.parser.parse_exceptions.MissingBelResource` (*line_number*, *line*, *args)

Raised when a missing resource is encountered

exception `pybel.parser.parse_exceptions.InvalidMetadataException` (*line_number*, *line*, *position*, *key*, *value*)

Raised when an incorrect document metadata key is used. Valid document metadata keys are:

- Authors
- ContactInfo
- Copyright
- Description
- Disclaimer
- Licenses
- Name
- Version

See also:

BEL specification on the [properties section](#)

exception `pybel.parser.parse_exceptions.MissingMetadataException` (*key*)

Raised when a BEL Script is missing critical metadata.

exception `pybel.parser.parse_exceptions.InvalidCitationLengthException` (*line_number*,
line, *po*-
sition,
**args*)

Base exception raised when the format for a citation is wrong.

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_exceptions.CitationTooShortException` (*line_number*,
line, *position*,
**args*)

Raised when a citation does not have the minimum of {type, name, reference}.

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_exceptions.CitationTooLongException` (*line_number*,
line, *position*,
**args*)

Raised when a citation has more than the allowed entries, {type, name, reference, date, authors, comments}.

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_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()`.

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_exceptions.MissingSupportWarning` (*line_number*, *line*, *po-*
sition, **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.

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_exceptions.InvalidCitationType` (*line_number*, *line*, *posi-*
tion, *citation_type*)

Raise when a citation is set with an incorrect type. Valid citation types include:

- Book
- PubMed
- Journal
- Online Resource
- URL
- DOI
- Other

See also:

[OpenBEL wiki on citations](#)

exception `pybel.parser.parse_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.

exception `pybel.parser.parse_exceptions.MalformedTranslocationWarning` (*line_number*, *line*, *position*, *tokens*)

Raised when there is a translocation statement without location information.

exception `pybel.parser.parse_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.

exception `pybel.parser.parse_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.

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
- **args** – Additional arguments to supply to the super class

exception `pybel.parser.parse_exceptions.LexicographyWarning`

Raised when encountering improper capitalization of namespace/annotation names.

exception `pybel.parser.parse_exceptions.InvalidFunctionSemantic` (*line_number*, *line*, *position*, *function*, *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)

exception `pybel.parser.parse_exceptions.RelabelWarning` (*line_number*, *line*, *position*, *node*, *old_label*, *new_label*)

Raised when a node is relabeled

Constants

PyBEL Constants

This module maintains the strings used throughout the PyBEL codebase to promote consistency.

Configuration Loading

By default, PyBEL loads its configuration from `~/.config/pybel/config.json`. This json is stored in the object `pybel.constants.config`.

`pybel.constants.GOCC_LATEST = 'https://arty.scai.fraunhofer.de/artifactory/bel/namespace/go-cellular-component/go-cel`

GOCC is the only namespace that needs to be stored because translocations use some of its values by default

`pybel.constants.PYBEL_CONNECTION = 'PYBEL_CONNECTION'`

The environment variable that contains the default SQL connection information for the PyBEL cache

`pybel.constants.PYBEL_DIR = '/home/docs/pybel'`

The default directory where PyBEL files, including logs and the default cache, are stored. Created if not exists.

`pybel.constants.PYBEL_LOG_DIR = '/home/docs/pybel/logs'`

The default directory where PyBEL logs are stored

`pybel.constants.PYBEL_DATA_DIR = '/home/docs/pybel/data'`

The default directory where PyBEL data are stored

`pybel.constants.DEFAULT_CACHE_LOCATION = '/home/docs/pybel/data/pybel_cache.db'`

The default cache location is `~/pybel/data/pybel_cache.db`

`pybel.constants.DEFAULT_CACHE_CONNECTION = 'sqlite:///home/docs/pybel/data/pybel_cache.db'`

The default cache connection string uses sqlite.

`pybel.constants.config = {'PYBEL_CONNECTION': 'sqlite:///home/docs/pybel/data/pybel_cache.db'}`

The global configuration for PyBEL is stored here. By default, it loads from `~/.config/pybel/config.json`

`pybel.constants.get_cache_connection()`

Returns the default cache connection string

`pybel.constants.BEL_DEFAULT_NAMESPACE = 'bel'`

The default namespace given to entities in the BEL language

`pybel.constants.CITATION_TYPES = {'DOI', 'Other', 'Journal', 'Book', 'PubMed', 'Online Resource', 'URL'}`

The valid citation types .. seealso: <https://wiki.openbel.org/display/BELNA/Citation>

`pybel.constants.NAMESPACE_DOMAIN_TYPES = {'Gene and Gene Products', 'Other', 'Chemical', 'BiologicalProcess'}`

The valid namespace types .. seealso: <https://wiki.openbel.org/display/BELNA/Custom+Namespaces>

`pybel.constants.CITATION_TYPE = 'type'`

Represents the key for the citation type in a citation dictionary

`pybel.constants.CITATION_NAME = 'name'`

Represents the key for the citation name in a citation dictionary

`pybel.constants.CITATION_REFERENCE = 'reference'`

Represents the key for the citation reference in a citation dictionary

`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_COMMENTS = 'comments'`

Represents the key for the citation comment in a citation dictionary

`pybel.constants.CITATION_ENTRIES = ('type', 'name', 'reference', 'date', 'authors', 'comments')`

Represents the ordering of the citation entries in a control statement (SET Citation = ...)

`pybel.constants.FUNCTION = 'function'`

The node data key specifying the node's function (e.g. *GENE*, *MIRNA*, *BIOPROCESS*, etc.)

`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.DESCRPTION = 'description'`

The key specifying an optional description for the node

`pybel.constants.FUSION = 'fusion'`

The node data key specifying a fusion dictionary, containing *PARTNER_3P*, *PARTNER_5P*,

`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.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.DIRTY = 'dirty'`

Used as a namespace when none is given when lenient parsing mode is turned on. Not recommended!

`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.PROTEIN = 'Protein'`

Represents the BEL abundance, `proteinAbundance()`

`pybel.constants.MIRNA = 'miRNA'`

Represents the BEL abundance, `microRNAAbundance()`

`pybel.constants.ABUNDANCE = 'Abundance'`
Represents the BEL abundance, abundance()

`pybel.constants.BIOPROCESS = 'BiologicalProcess'`
Represents the BEL function, biologicalProcess()

`pybel.constants.PATHOLOGY = 'Pathology'`
Represents the BEL function, pathology()

`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.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.SUBJECT = 'subject'`
The key for an internal edge data dictionary for the subject modifier dictionary

`pybel.constants.OBJECT = 'object'`
The key for an internal edge data dictionary for the object modifier dictionary

`pybel.constants.LINE = 'line'`
The key or an internal edge data dictionary for the line number

`pybel.constants.ID = 'id'`
The key representing the hash of the other

`pybel.constants.PYBEL_EDGE_DATA_KEYS = {'object', 'citation', 'relation', 'subject', 'evidence', 'annotations'}`
The group of all BEL-provided keys for edge data dictionaries

`pybel.constants.PYBEL_EDGE_ALL_KEYS = {'id', 'object', 'citation', 'relation', 'line', 'subject', 'evidence', 'annotations'}`
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_COMPONENT = 'hasComponent'`
A BEL relationship

`pybel.constants.HAS_VARIANT = 'hasVariant'`
A BEL relationship

`pybel.constants.HAS_MEMBER = 'hasMember'`
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.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.CAUSAL_INCREASE_RELATIONS = {'increases', 'directlyIncreases'}`
 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_RELATIONS = {'increases', 'directlyDecreases', 'directlyIncreases', 'decreases'}`
 A set of all causal relationships

`pybel.constants.TWO_WAY_RELATIONS = {'negativeCorrelation', 'analogousTo', 'association', 'orthologous', 'equivalent'}`
 A set of all relationships that are inherently directionless, and are therefore added to the graph twice

`pybel.constants.CORRELATIVE_RELATIONS = {'negativeCorrelation', 'positiveCorrelation'}`
 A set of all correlative relationships

`pybel.constants.unqualified_edges = ['hasReactant', 'hasProduct', 'hasComponent', 'hasVariant', 'transcribedTo']`
 A list of relationship types that don't require annotations or evidence This must be maintained as a list, since the `unqualified_edge_code` is calculated based on the order and needs to be consistent

`pybel.constants.unqualified_edge_code = {'isA': -8, 'hasMember': -7, 'hasVariant': -4, 'hasComponent': -3, 'transcribedTo': -2}`
 Unqualified edges are given negative keys since the standard NetworkX edge key factory starts at 0 and counts up

`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 = {'ContactInfo': 'contact', 'Authors': 'authors', 'Name': 'name', 'Licenses': 'licenses'}`
 Provides a mapping from BEL language keywords to internal PyBEL strings

`pybel.constants.METADATA_INSERT_KEYS = {'contact', 'name', 'version', 'authors', 'description', 'licenses', 'copyright'}`
 The keys to use when inserting a graph to the cache

`pybel.constants.INVERSE_DOCUMENT_KEYS = {'project': 'Project', 'licenses': 'Licenses', 'contact': 'ContactInfo', 'description': 'Description'}`
 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 = {'O': {'Pathology'}, 'B': {'Pathology', 'BiologicalProcess'}, '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>

BEL Language

This module contains mappings between PyBEL's internal constants and BEL language keywords

`pybel.parser.language.abundance_labels = {'bp': 'BiologicalProcess', 'rnaAbundance': 'RNA', 'path': 'Pathology'}`
Provides a mapping from BEL terms to PyBEL internal constants

`pybel.parser.language.aa_placeholder = "X"`
In biological literature, the X is used to denote a truncation. Text mining efforts often encode X as an amino acid, for which we will throw an error using `handle_aa_placeholder()`

`pybel.parser.language.handle_aa_placeholder` (*line, position, tokens*)

Raises an exception when encountering a placeholder amino acid, X

`pybel.parser.language.pmod_namespace` = {'ADP-ribosylation': 'ADPRib', 'hydroxylation': 'Hy', 'neddylation': 'Ne

A dictionary of default protein modifications to their preferred value

`pybel.parser.language.pmod_legacy_labels` = {'U': 'Ub', 'S': 'Sumo', 'O': 'Ox', 'R': 'ADPRib', 'P': 'Ph', 'F': 'Fa

A dictionary of legacy (BEL 1.0) default namespace protein modifications to their BEL 2.0 preferred value

`pybel.parser.language.gmod_namespace` = {'Me': 'Me', 'M': 'Me', 'methylation': 'Me'}

A dictionary of default gene modifications. This is a PyBEL variant to the BEL specification.

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 1.0 statements as BEL 2.0. 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

Metadata Parser

```
class pybel.parser.parse_metadata.MetadataParser(manager, namespace_dict=None,
                                                annotation_dict=None, namespace_regex=None,
                                                annotations_regex=None, default_namespace=None)
```

A parser for the document and definitions section of a BEL document.

See also:

BEL 1.0 Specification for the [DEFINE](#) keyword

Parameters

- **manager** (`pybel.manager.cache.CacheManager`) – A cache manager
- **namespace_dict** (*dict*) – A dictionary of pre-loaded, enumerated namespaces from {namespace keyword: set of valid values}
- **annotation_dict** (*dict*) – A dictionary of pre-loaded, enumerated annotations from {annotation keyword: set of valid values}
- **namespace_regex** (*dict*) – A dictionary of pre-loaded, regular expression namespaces from {namespace keyword: regex string}
- **annotations_regex** (*dict*) – A dictionary of pre-loaded, regular expression annotations from {annotation keyword: regex string}
- **default_namespace** (*set of str*) – A set of strings that can be used without a namespace

manager = None

This metadata parser's internal definition cache manager

namespace_dict = None
A dictionary of cached {namespace keyword: set of values}

annotations_dict = None
A dictionary of cached {annotation keyword: set of values}

namespace_regex = None
A dictionary of {namespace keyword: regular expression string}

namespace_regex_compiled = None
A dictionary of {namespace keyword: compiled regular expression}

default_namespace = None
A set of names that can be used without a namespace

annotations_regex = None
A dictionary of {annotation keyword: regular expression string}

annotations_regex_compiled = None
A dictionary of {annotation keyword: compiled regular expression}

document_metadata = None
A dictionary containing the document metadata

namespace_url_dict = None
A dictionary from {namespace keyword: BEL namespace URL}

namespace_owl_dict = None
A dictionary from {namespace keyword: OWL namespace URL}

annotation_url_dict = None
A dictionary from {annotation keyword: BEL annotation URL}

annotations_owl_dict = None
A dictionary from {annotation keyword: OWL annotation URL}

annotation_lists = None
A set of annotation keywords that are defined ad-hoc in the BEL script

handle_document (*line, position, tokens*)
Handles statements like SET DOCUMENT X = "Y"

handle_namespace_url (*line, position, tokens*)
Handles statements like DEFINE NAMESPACE X AS URL "Y"

handle_namespace_owl (*line, position, tokens*)
Handles statements like DEFINE NAMESPACE X AS OWL "Y"

handle_namespace_pattern (*line, position, tokens*)
Handles statements like DEFINE NAMESPACE X AS PATTERN "Y"

handle_annotation_owl (*line, position, tokens*)
Handles statements like DEFINE ANNOTATION X AS OWL "Y"

handle_annotations_url (*line, position, tokens*)
Handles statements like DEFINE ANNOTATION X AS URL "Y"

handle_annotation_list (*line, position, tokens*)
Handles statements like DEFINE ANNOTATION X AS LIST {"Y", "Z", ...}

handle_annotation_pattern (*line, position, tokens*)
Handles statements like DEFINE ANNOTATION X AS PATTERN "Y"

has_enumerated_annotation (*annotation*)

Checks if this annotation is defined by an enumeration

has_regex_annotation (*annotation*)

Checks if this annotation is defined by a regular expression

has_annotation (*annotation*)

Checks if this annotation is defined

has_enumerated_namespace (*namespace*)

Checks if this namespace is defined by an enumeration

has_regex_namespace (*namespace*)

Checks if this namespace is defined by a regular expression

has_namespace (*namespace*)

Checks if this namespace is defined

check_version (*line, position, s*)

Checks that a version string is valid for BEL documents, meaning it's either in the YYYYMMDD or semantic version format

Control Parser

class `pybel.parser.parse_control.ControlParser` (*annotation_dicts=None, annotation_regex=None, citation_clearing=True*)

A parser for BEL control statements

See also:

BEL 1.0 specification on [control records](#)

Parameters

- **set[str]** **annotation_dicts** (*dict[str, ...]*) – A dictionary of {annotation: set of valid values} for parsing
- **str** **annotation_regex** (*dict[str, ...]*) – A dictionary of {annotation: regular expression string}
- **bool** **citation_clearing** (*bool*) – Should SET Citation statements clear evidence and all annotations?

handle_annotation_key (*line, position, tokens*)

Called on all annotation keys before parsing to validate that it's either enumerated or as a regex

get_annotations ()

Returns The currently stored BEL annotations

Return type `dict`

clear_citation ()

Clears the citation. Additionally, if citation clearing is enabled, clears the evidence and annotations.

clear ()

Clears the `statement_group`, `citation`, `evidence`, and `annotations`

Identifier Parser

```
class pybel.parser.parse_identifier.IdentifierParser(namespace_dict=None,
                                                  namespace_regex=None,    de-
                                                  fault_namespace=None,    al-
                                                  low_naked_names=False)
```

A parser for identifiers in the form of namespace:name. Can be made more lenient when given a default namespace or enabling the use of naked names

Parameters

- **namespace_dict** (*dict[str, set[str]]*) – A dictionary of {namespace: set of names}
- **namespace_regex** (*dict[str, str]*) – A dictionary of {namespace: regular expression string} to compile
- **default_namespace** (*set[str]*) – A set of strings that can be used without a namespace
- **allow_naked_names** (*bool*) – If true, turn off naked namespace failures

namespace_dict = None

A dictionary of cached {namespace keyword: set of values}

namespace_regex = None

A dictionary of {namespace keyword: regular expression string}

namespace_regex_compiled = None

A dictionary of {namespace keyword: compiled regular expression}

has_enumerated_namespace (*namespace*)

Checks that the namespace has been defined by an enumeration

has_regex_namespace (*namespace*)

Checks that the namespace has been defined by a regular expression

has_namespace (*namespace*)

Checks that the namespace has either been defined by an enumeration or a regular expression

has_enumerated_namespace_name (*namespace, name*)

Checks that the namespace is defined by an enumeration and that the name is a member

has_regex_namespace_name (*namespace, name*)

Checks that the namespace is defined as a regular expression and the name matches it

BEL Parser

```
class pybel.parser.parse_bel.BelParser(graph, namespace_dict=None, annotation_dict=None,
                                       namespace_regex=None,    annotation_regex=None,
                                       allow_naked_names=False,  allow_nested=False,
                                       allow_unqualified_translocations=False, citation_clearing=True,
                                       warn_on_singleton=True,
                                       no_identifier_validation=False, autostreamline=True)
```

Build a parser backed by a given dictionary of namespaces

Parameters

- **graph** ([BELGraph](#)) – The BEL Graph to use to store the network

- **namespace_dict** (*dict[str, set[str]]*) – A dictionary of {namespace: set of members}. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **annotation_dict** (*dict[str, set[str]]*) – A dictionary of {annotation: set of values}. Delegated to `pybel.parser.ControlParser`
- **namespace_regex** (*dict[str, str]*) – A dictionary of {namespace: regular expression strings}. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **annotation_regex** (*dict[str, str]*) – A dictionary of {annotation: regular expression strings}. Delegated to `pybel.parser.ControlParser`
- **allow_naked_names** (*bool*) – If true, turn off naked namespace failures. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **allow_nested** (*bool*) – If true, turn off nested statement failures. Delegated to `pybel.parser.parse_identifier.IdentifierParser`
- **allow_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`
- **warn_on_singleton** (*bool*) – Should the parser thorough warnings on singletons? Only disable this if you're sure your BEL Script is syntactically and semantically valid.
- **autostreamline** (*bool*) – Should the parser be streamlined on instantiation?

pmod = None

2.2.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_protein_modifications

variant = None

2.2.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_variant_var

fragment = None

2.2.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_proteolytic_fragments

location = None

2.2.4 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_cellular_location

psub = None

2.2.X DEPRECATED http://openbel.org/language/web/version_1.0/bel_specification_version_1.0.html#_amino_acid_substitutions

gsub = None

2.2.X DEPRECATED http://openbel.org/language/web/version_1.0/bel_specification_version_1.0.html#_sequence_variations

t trunc = None

DEPRECATED http://openbel.org/language/web/version_1.0/bel_specification_version_1.0.html#_truncated_proteins

gmod = None

PyBEL BEL Specification variant

fusion = None

2.6.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_fusion_fus

general_abundance = None2.1.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XcomplexA**gene = None**2.1.4 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XgeneA**mirna = None**2.1.5 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XmicroRNAA**protein = None**2.1.6 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XproteinA**rna = None**2.1.7 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XrnaA**complex_singleton = None**2.1.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XcomplexA**composite_abundance = None**2.1.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XcompositeA**molecular_activity = None**2.4.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XmolecularA**biological_process = None**2.3.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_biologicalprocess_bp**pathology = None**2.3.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_pathology_path**activity = None**2.3.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#Xactivity**translocation = None**2.5.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_translocations**degradation = None**2.5.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_degradation_deg**reactants = None**2.5.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_reaction_rxn**increases_tag = None**3.1.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#Xincreases**directly_increases_tag = None**3.1.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XdIncreases**decreases_tag = None**3.1.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#Xdecreases**directly_decreases_tag = None**3.1.4 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XdDecreases**analogous_tag = None**3.5.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_analogous**causes_no_change_tag = None**3.1.6 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#Xcnc**regulates_tag = None**3.1.7 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_regulates_reg

negative_correlation_tag = None

3.2.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XnegCor

positive_correlation_tag = None

3.2.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#XposCor

association_tag = None

3.2.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#Xassociation

orthologous_tag = None

3.3.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_orthologous

is_a_tag = None

3.4.5 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_isa

equivalent_tag = None

PyBEL Variant

rate_limit_tag = None

3.1.5 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_ratelimitingstepof

subprocess_of_tag = None

3.4.6 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_subprocessof

transcribed_tag = None

3.3.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_transcribedto

translated_tag = None

3.3.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_translatedto

has_member_tag = None

3.4.1 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_hasmember

abundance_list = None

3.4.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_hasmembers

biomarker_tag = None

3.5.2 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_biomarkerfor

prognostic_biomarker_tag = None

3.5.3 http://openbel.org/language/web/version_2.0/bel_specification_version_2.0.html#_prognosticbiomarkerfor

causal_relation_tags = None

3.1 Causal Relationships - nested. Explicitly not supported because of ambiguity

get_annotations ()

Get current annotations in this parser

Return type dict

clear ()

Clears the graph and all control parser data (current citation, annotations, and statement group)

handle_term_singleton (line, position, tokens)

This function wraps `BelParser.handle_term()` but is only used for top-level parsing of `bel_terms`. This is done solely to keep track of if a graph has any singletons

ensure_node (tokens)

Turns parsed tokens into canonical node name and makes sure its in the graph

Returns the canonical name of the node

Return type `str`

Cache

Cache Manager

class `pybel.manager.base_cache.BaseCacheManager` (*connection=None, echo=False*)
Creates 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** (*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_LOCATION`
- **echo** (*bool*) – Turn on echoing sql

sessionmaker = None

A SQLAlchemy session maker

session = None

A SQLAlchemy session object

create_all (*checkfirst=True*)

Creates the PyBEL cache's database and tables

drop_database ()

Drops all data, tables, and databases for the PyBEL cache

class `pybel.manager.cache.CacheManager` (**args, **kwargs*)

The definition cache manager takes care of storing BEL namespace and annotation files for later use. It uses SQLite by default for speed and lightness, but any database can be used with its SQLAlchemy interface.

Database 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

url

Source url of the given namespace definition file (.belns)

keyword

Keyword that is used in a BEL file to identify a specific namespace

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

version

Version of the namespace

created

DateTime of the creation of the namespace definition file

query_url

URL that can be used to query the namespace (eternally from PyBEL)

author

The author of the namespace

license

License information

contact

Contact information

data

Returns the table entry as a dictionary without the SQLAlchemy instance information.

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

encoding

The biological entity types for which this name is valid

data

Describes the namespaceEntry as dictionary of Namespace-Keyword and Name.

to_json()

Enables json serialization for the class this method is defined in.

class `pybel.manager.models.NamespaceEntryEquivalence(**kwargs)`

Represents the equivalence classes between entities

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.Annotation(**kwargs)`

Represents a BEL Annotation

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

url

Source url of the given annotation definition file (.belanno)

keyword

Keyword that is used in a BEL file to identify a specific annotation

type

Annotation type

description

Optional short description of the given annotation

version

Version of the annotation

created

DateTime of the creation of the given annotation definition

name

Name of the annotation definition

author

Author information

license

License information

contact

Contact information

class `pybel.manager.models.AnnotationEntry(**kwargs)`

Represents a value within a BEL Annotation

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 annotation definition file

data

Describes the annotationEntry as dictionary of Annotation-Keyword and Annotation-Name.

to_json()

Enables json serialization for the class this method is defined in.

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 information extracted from the underlying BEL file

description

Descriptive text extracted from the BEL file

copyright

Copyright information

disclaimer

Disclaimer information

licenses

License information

blob

A pickled version of this network

as_bel()

Gets this network and loads it into a BELGraph

Return type *BELGraph*

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

namespacePattern

Contains regex pattern for value identification.

is_variant

Identifies weather or not the given node is a variant

fusion

Identifies weather or not the given node is a fusion

bel

Valid BEL term that represents the given node

to_json()

Enables json serialization for the class this method is defined in.

class `pybel.manager.models.Modification` (**kwargs)

The modifications that are present in the network 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.

modType

Type of the stored modification e.g. Fusion

modNamespace

Namespace for the modification name

modName

Name of the given modification (used for pmod or gmod)

aminoA

Three letter amino acid code

position

Position

data

Recreates a `is_variant` dictionary for `BELGraph`

Returns Dictionary that describes a variant or a fusion.

Return type `dict`

to_json()

Enables json serialization for the class this method is defined in.

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.

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.

type

Type of the stored publication e.g. PubMed

name

Title of the publication

reference

Reference identifier of the publication e.g. PubMed_ID

date

Publication date

data

Creates a citation dictionary that is used to recreate the edge data dictionary of a BELGraph.

Returns Citation dictionary for the recreation of a BELGraph.

Return type `dict`

to_json ()

Enables json serialization for the class this method is defined in.

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 that is cited from a given publication

data

Creates a dictionary that is used to recreate the edge data dictionary for a BELGraph.

Returns Dictionary containing citation and evidence for a BELGraph edge.

Return type `dict`

to_json ()

Enables json serialization for the class this method is defined in.

class `pybel.manager.models.Edge` (**kwargs)

Relationships are represented in this table. It shows the nodes that are in a relation to each other and provides information about the context of the relation by referring to the annotation, property and evidence tables.

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

data

Creates a dictionary of one BEL Edge that can be used to create an edge in a BELGraph.

Returns Dictionary that contains information about an edge of a BELGraph. Including participants and edge data information.

Return type `dict`

class `pybel.manager.models.Property` (**kwargs)

The property table contains additional information that is used to describe the context of a relation.

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.

participant

Identifies which participant of the edge is affected by the given property

modifier

The modifier to the corresponding participant

effectNamespace

Optional namespace that defines modifier value

effectName

Value for specific modifiers e.g. Activity

relativeKey

Relative key of effect e.g. `to_tloc` or `from_tloc`

propValue

Value of the effect

data

Creates a property dict that is used to recreate an edge dictionary for a BELGraph.

Returns Property dictionary of an edge that is participant (sub/obj) related.

Return type `dict`

to_json()

Enables json serialization for the class this method is defined in.

Utilities

Some utilities that are used throughout the software are explained here:

General Utilities

`pybel.utils.download(url)`

Uses requests to download an URL, maybe from a file

`pybel.utils.parse_bel_resource(lines)`

Parses a BEL config (BELNS, BELANNO, or BELEQ) file from the given line iterator over the file

Parameters `lines` (*iter[str]*) – An iterable over the lines in a BEL config file

Returns A config-style dictionary representing the BEL config file

Return type `dict`

`pybel.utils.is_url(s)`

Checks if a string is a valid URL

Parameters `s` (*str*) – An input string

Returns Is the string a valid URL?

Return type `bool`

`pybel.utils.get_bel_resource(location)`

Loads/downloads and parses a config file from the given url or file path

Parameters `location` (*str*) – The URL or file path to a BELNS, BELANNO, or BELEQ file to download and parse

Returns A config-style dictionary representing the BEL config file

Return type `dict`

`pybel.utils.expand_dict(flat_dict, sep='_')`

Expands a flattened dictionary

Parameters

- **flat_dict** (*dict*) – a nested dictionary that has been flattened so the keys are composite
- **sep** (*str*) – the separator between concatenated keys

`pybel.utils.flatten_dict(d, parent_key='', sep='_')`

Flattens a nested dictionary.

Parameters

- **d** (*dict* or *MutableMapping*) – A nested dictionary
- **parent_key** (*str*) – The parent's key. This is a value for tail recursion, so don't set it yourself.
- **sep** (*str*) – The separator used between dictionary levels

See also:

<http://stackoverflow.com/a/6027615>

`pybel.utils.flatten_graph_data(graph)`

Returns a new graph with flattened edge data dictionaries.

Parameters `graph` (*nx.MultiDiGraph*) – A graph with nested edge data dictionaries

Returns A graph with flattened edge data dictionaries

Return type `nx.MultiDiGraph`

`pybel.utils.list2tuple(l)`

Recursively converts a nested list to a nested tuple

`pybel.utils.get_version()`

Gets the current PyBEL version

Returns The current PyBEL version

Return type `str`

`pybel.utils.tokenize_version` (*version_string*)
 Tokenizes a version string to a tuple. Truncates qualifiers like `-dev`.

Parameters `version_string` (*str*) – A version string

Returns A tuple representing the version string

Return type `tuple`

```
>>> tokenize_version('0.1.2-dev')
(0, 1, 2)
```

`pybel.utils.citation_dict_to_tuple` (*citation*)
 Convert the `d[CITATION]` entry in an edge data dictionary to a tuple

`pybel.utils.flatten_citation` (*citation*)
 Flattens a citation dict, from the `d[CITATION]` entry in an edge data dictionary

`pybel.utils.ensure_quotes` (*s*)
 Quote a string that isn't solely alphanumeric

Parameters `s` (*str*) – a string

Return type `str`

`pybel.utils.valid_date` (*s*)
 Checks that a string represents a valid date in ISO 8601 format `YYYY-MM-DD`

Return type `bool`

`pybel.utils.valid_date_version` (*s*)
 Checks that the string is a valid date versions string

`pybel.utils.parse_datetime` (*s*)
 Tries to parse a datetime object from a standard datetime format or date format

Parameters `s` (*str*) – A string representing a date or datetime

Returns A parsed date object

Return type `datetime.date`

`pybel.utils.hash_node` (*node*)
 Converts a PyBEL node tuple to a hash

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

Returns A hashed version of the node tuple using md5 hash of the binary pickle dump

Return type `str`

`pybel.utils.extract_pybel_data` (*data*)
 Extracts only the PyBEL-specific data from the given edge data dictionary

Parameters `data` (*dict*) – An edge data dictionary

Return type `dict`

`pybel.utils.edge_to_tuple` (*u, v, k, data*)
 Converts an edge to tuple

Parameters

- `u` (*tuple*) – The source BEL node
- `v` (*tuple*) – The target BEL node

- **data** (*dict*) – The edge’s data dictionary

Returns A tuple that can be hashed representing this edge. Makes no promises to its structure.

`pybel.utils.hash_edge(u, v, k, d)`

Converts an edge tuple to a hash

Returns A hashed version of the edge tuple using md5 hash of the binary pickle dump of u, v, and the json dump of d

Return type `str`

`pybel.utils.subdict_matches(target, query, partial_match=True)`

Checks if all the keys in the query dict are in the target dict, and that their values match

1. Checks that all keys in the query dict are in the target dict

2. **Matches the values of the keys in the query dict**

- (a) If the value is a string, then must match exactly
- (b) If the value is a set/list/tuple, then will match any of them
- (c) If the value is a dict, then recursively check if that subdict matches

Parameters

- **target** (*dict*) – The dictionary to search
- **query** (*dict*) – A query dict with keys to match
- **partial_match** (*bool*) – Should the query values be used as partial or exact matches? Defaults to `True`.

Returns if all keys in b are in target_dict and their values match

Return type `bool`

Parser Utilities

`pybel.parser.utils.any_subdict_matches(dict_of_dicts, query_dict)`

Checks if dictionary target_dict matches one of the subdictionaries of a

Parameters

- **dict_of_dicts** – dictionary of dictionaries
- **query_dict** – dictionary

Returns if dictionary target_dict matches one of the subdictionaries of a

Return type `bool`

`pybel.parser.utils.cartesian_dictionary(d)`

takes a dictionary of sets and provides subdicts

Parameters **d** (*dict*) – a dictionary of sets

Return type `list`

`pybel.parser.utils.is_int(s)`

Determines if an object can be cast to an int

Parameters **s** – any object

Returns true if argument can be cast to an int:

Return type `bool`

`pybel.parser.utils.nest` (**content*)

Defines a delimited list by enumerating each element of the list

`pybel.parser.utils.one_of_tags` (*tags, canonical_tag, identifier*)

This is a convenience method for defining the tags usable in the `BelParser`. For example, statements like `g(HGNC:SNCA)` can be expressed also as `geneAbundance(HGNC:SNCA)`. The language must define multiple different tags that get normalized to the same thing.

Parameters

- **tags** (*list of str*) – a list of strings that are the tags for a function. For example, `['g', 'geneAbundance']` for the abundance of a gene
- **canonical_tag** (*str*) – the preferred tag name. Does not have to be one of the tags. For example, `'GeneAbundance'` (note capitalization) is used for the abundance of a gene
- **identifier** – this is the key under which the value for this tag is put in the PyParsing framework.

Returns a PyParsing `ParseElement`

Return type `ParseElement`

`pybel.parser.utils.triple` (*subject, relation, obj*)

Builds a simple triple in PyParsing that has a `subject relation object` format

Roadmap

Current Features

- Easy to install with pip
- Easy to use, with most common functions at top package level
- Good documentation and tutorials
- Extensible API, allowing for bioinformaticians to develop their own algorithms and own their data, instead of using a complicated plugin in Cytoscape with multiple dependencies and little ability for export or customization
- Caching system for downloaded namespace and annotations
- Fully BEL 1.0 and BEL 2.0 support
- Node and edge filtering API
- Usage of OWL documents as namespaces

Issues

- 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.
- 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.

- 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.

Roadmap

Performance

- Parallelization of BEL tokenization

Knowledge Assembly

- Integration of prior knowledge, such orthology between genes/proteins and resolution of protein complex names
- Integration of heirarchical information, insertion of missing complex members and protein family heirarchies
- Leverage of ontologies in the OWL format for disease-specific entities and annotations

Post-Processing Tools

- Implementation of Reverse Causal Reasoning and Network Perturbation Amplitude
- Distributed BEL document infrastructure
- Semantic Diff tool

Data Interchange

Another format, Biological Pathways Expression Language (BioPax), was developed to integrate pathway information from many disparate databases. Like BEL, it can describe metabolic pathways and molecular interactions, but it excels in signaling pathways, gene regulatory networks, and genetic interactions. Systems Biology Markup Language (SBML) is the third common format that provides a more general framework for building quantitative and temporal models.

Data locked away in other formats such as BioPax and SBML cannot be accessed by PyBEL currently. Development of knowledge assemblers, like INDRA, provide support for import of many formats. PyBEL will enable the import of BEL documents much more quickly, and ultimately enable the export of SMBL. In the future, it would also be useful to develop additional interchange tools for BioPax to BEL, but we recognize that this is a large task that will be limited by the expressibility of each language and the difficult development of a two-way mapping.

Technology

This page is meant to describe the development stack for PyBEL, and should be a useful introduction for contributors.

Versioning

PyBEL is kept under version control on GitHub. This allows for changes in the software to be tracked over time, and for tight integration of the management aspect of software development. 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.

Testing in PyBEL

PyBEL is written with extensive unit testing and integration testing. Whenever possible, PyBEL prefers to practice test-driven development. 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. Some functionality, such as the `mock` module, are only available as default in Python 3, so backports must be used for testing in Python 2

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.

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

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 `py.test` 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.

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.

Code Coverage

After building, Travis sends code coverage results to 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.

Distribution

Versioning

PyBEL uses semantic versioning. In general, the project's version string will have 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.

Deployment

Code for PyBEL is open-source on GitHub, but it is also distributed on the PyPI (pronounced Py-Pee-Eye) server. 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.

CHAPTER 2

Indices and Tables

- genindex
- modindex
- search

p

pybel, 3
pybel.constants, 39
pybel.exceptions, 34
pybel.io, 22
pybel.io.cx, 27
pybel.io.gpickle, 24
pybel.io.jgif, 26
pybel.io.ndex_utils, 30
pybel.io.neo4j, 29
pybel.io.nodelink, 25
pybel.manager.database_io, 29
pybel.manager.models, 53
pybel.parser.language, 45
pybel.parser.modifiers.fragment, 14
pybel.parser.modifiers.fusion, 17
pybel.parser.modifiers.gene_modification,
15
pybel.parser.modifiers.gene_substitution,
13
pybel.parser.modifiers.location, 20
pybel.parser.modifiers.protein_modification,
16
pybel.parser.modifiers.protein_substitution,
13
pybel.parser.modifiers.truncation, 14
pybel.parser.modifiers.variant, 12
pybel.parser.parse_exceptions, 34
pybel.parser.utils, 62
pybel.struct, 10
pybel.utils, 59

A

aa_placeholder (in module pybel.parser.language), 45
 ABUNDANCE (in module pybel.constants), 41
 abundance_labels (in module pybel.parser.language), 45
 abundance_list (pybel.parser.parse_bel.BelParser attribute), 52
 activity (pybel.parser.parse_bel.BelParser attribute), 51
 add_simple_node() (pybel.BELGraph method), 11
 add_unqualified_edge() (pybel.BELGraph method), 11
 aminoA (pybel.manager.models.Modification attribute), 57
 analogous_tag (pybel.parser.parse_bel.BelParser attribute), 51
 ANALOGOUS_TO (in module pybel.constants), 43
 Annotation (class in pybel.manager.models), 55
 annotation_list (pybel.BELGraph attribute), 11
 annotation_lists (pybel.parser.parse_metadata.MetadataParser attribute), 47
 annotation_owl (pybel.BELGraph attribute), 11
 annotation_pattern (pybel.BELGraph attribute), 11
 annotation_url (pybel.BELGraph attribute), 11
 annotation_url_dict (pybel.parser.parse_metadata.MetadataParser attribute), 47
 AnnotationEntry (class in pybel.manager.models), 55
 AnnotationIdentifierWarning, 36
 ANNOTATIONS (in module pybel.constants), 42
 annotations_dict (pybel.parser.parse_metadata.MetadataParser attribute), 47
 annotations_owl_dict (pybel.parser.parse_metadata.MetadataParser attribute), 47
 annotations_regex (pybel.parser.parse_metadata.MetadataParser attribute), 47
 annotations_regex_compiled (pybel.parser.parse_metadata.MetadataParser attribute), 47
 AnnotationWarning, 36
 any_subdict_matches() (in module pybel.parser.utils), 62

as_bel() (pybel.manager.models.Network method), 56
 ASSOCIATION (in module pybel.constants), 43
 association_tag (pybel.parser.parse_bel.BelParser attribute), 52
 Author (class in pybel.manager.models), 57
 author (pybel.manager.models.Annotation attribute), 55
 author (pybel.manager.models.Namespace attribute), 54
 authors (pybel.manager.models.Network attribute), 56

B

Base (class in pybel.manager.models), 53
 BaseCacheManager (class in pybel.manager.base_cache), 53
 bel (pybel.manager.models.Edge attribute), 58
 bel (pybel.manager.models.Node attribute), 57
 BEL_DEFAULT_NAMESPACE (in module pybel.constants), 40
 BELGraph (class in pybel), 10
 belns_encodings (in module pybel.constants), 45
 BelParser (class in pybel.parser.parse_bel), 49
 biological_process (pybel.parser.parse_bel.BelParser attribute), 51
 BIOMARKER_FOR (in module pybel.constants), 43
 biomarker_tag (pybel.parser.parse_bel.BelParser attribute), 52
 BIOPROCESS (in module pybel.constants), 42
 blob (pybel.manager.models.Network attribute), 56

C

CacheManager (class in pybel.manager.cache), 53
 cartesian_dictionary() (in module pybel.parser.utils), 62
 CAUSAL_DECREASE_RELATIONS (in module pybel.constants), 43
 CAUSAL_INCREASE_RELATIONS (in module pybel.constants), 43
 causal_relation_tags (pybel.parser.parse_bel.BelParser attribute), 52
 CAUSAL_RELATIONS (in module pybel.constants), 43
 CAUSES_NO_CHANGE (in module pybel.constants), 43

- causes_no_change_tag (pybel.parser.parse_bel.BelParser attribute), 51
 - check_version() (pybel.parser.parse_metadata.MetadataParser method), 48
 - Citation (class in pybel.manager.models), 57
 - CITATION (in module pybel.constants), 42
 - CITATION_AUTHORS (in module pybel.constants), 40
 - CITATION_COMMENTS (in module pybel.constants), 40
 - CITATION_DATE (in module pybel.constants), 40
 - citation_dict_to_tuple() (in module pybel.utils), 61
 - CITATION_ENTRIES (in module pybel.constants), 40
 - CITATION_NAME (in module pybel.constants), 40
 - CITATION_REFERENCE (in module pybel.constants), 40
 - CITATION_TYPE (in module pybel.constants), 40
 - CITATION_TYPES (in module pybel.constants), 40
 - CitationTooLongException, 37
 - CitationTooShortException, 37
 - clear() (pybel.parser.parse_bel.BelParser method), 52
 - clear() (pybel.parser.parse_control.ControlParser method), 48
 - clear_citation() (pybel.parser.parse_control.ControlParser method), 48
 - COMPLEX (in module pybel.constants), 42
 - complex_singleton (pybel.parser.parse_bel.BelParser attribute), 51
 - COMPOSITE (in module pybel.constants), 42
 - composite_abundance (pybel.parser.parse_bel.BelParser attribute), 51
 - config (in module pybel.constants), 40
 - contact (pybel.manager.models.Annotation attribute), 55
 - contact (pybel.manager.models.Namespace attribute), 54
 - contact (pybel.manager.models.Network attribute), 56
 - ControlParser (class in pybel.parser.parse_control), 48
 - copyright (pybel.manager.models.Network attribute), 56
 - CORRELATIVE_RELATIONS (in module pybel.constants), 44
 - create_all() (pybel.manager.base_cache.BaseCacheManager method), 53
 - created (pybel.manager.models.Annotation attribute), 55
 - created (pybel.manager.models.Namespace attribute), 54
- ## D
- data (pybel.manager.models.AnnotationEntry attribute), 56
 - data (pybel.manager.models.Citation attribute), 58
 - data (pybel.manager.models.Edge attribute), 59
 - data (pybel.manager.models.Evidence attribute), 58
 - data (pybel.manager.models.Modification attribute), 57
 - data (pybel.manager.models.Namespace attribute), 54
 - data (pybel.manager.models.NamespaceEntry attribute), 55
 - data (pybel.manager.models.Property attribute), 59
- date (pybel.manager.models.Citation attribute), 58
 - DECREASES (in module pybel.constants), 43
 - decreases_tag (pybel.parser.parse_bel.BelParser attribute), 51
 - DEFAULT_CACHE_CONNECTION (in module pybel.constants), 40
 - DEFAULT_CACHE_LOCATION (in module pybel.constants), 40
 - default_namespace (pybel.parser.parse_metadata.MetadataParser attribute), 47
 - degradation (pybel.parser.parse_bel.BelParser attribute), 51
 - DESCRIPTION (in module pybel.constants), 41
 - description (pybel.BELGraph attribute), 11
 - description (pybel.manager.models.Annotation attribute), 55
 - description (pybel.manager.models.Namespace attribute), 54
 - description (pybel.manager.models.Network attribute), 56
 - DIRECTLY_DECREASES (in module pybel.constants), 43
 - directly_decreases_tag (pybel.parser.parse_bel.BelParser attribute), 51
 - DIRECTLY_INCREASES (in module pybel.constants), 43
 - directly_increases_tag (pybel.parser.parse_bel.BelParser attribute), 51
 - DIRTY (in module pybel.constants), 41
 - disclaimer (pybel.manager.models.Network attribute), 56
 - document (pybel.BELGraph attribute), 11
 - DOCUMENT_KEYS (in module pybel.constants), 44
 - document_metadata (pybel.parser.parse_metadata.MetadataParser attribute), 47
 - domain (pybel.manager.models.Namespace attribute), 54
 - download() (in module pybel.utils), 59
 - drop_database() (pybel.manager.base_cache.BaseCacheManager method), 53
- ## E
- Edge (class in pybel.manager.models), 58
 - edge_to_tuple() (in module pybel.utils), 61
 - effectName (pybel.manager.models.Property attribute), 59
 - effectNamespace (pybel.manager.models.Property attribute), 59
 - encoding (pybel.manager.models.NamespaceEntry attribute), 54
 - ensure_node() (pybel.parser.parse_bel.BelParser method), 52
 - ensure_quotes() (in module pybel.utils), 61
 - equivalent_tag (pybel.parser.parse_bel.BelParser attribute), 52

EQUIVALENT_TO (in module pybel.constants), 43
 Evidence (class in pybel.manager.models), 58
 EVIDENCE (in module pybel.constants), 42
 expand_dict() (in module pybel.utils), 60
 extract_pybel_data() (in module pybel.utils), 61

F

flatten_citation() (in module pybel.utils), 61
 flatten_dict() (in module pybel.utils), 60
 flatten_graph_data() (in module pybel.utils), 60
 FRAGMENT (in module pybel.constants), 41
 fragment (pybel.parser.parse_bel.BelParser attribute), 50
 FRAGMENT_DESCRIPTION (in module pybel.constants), 45
 FRAGMENT_MISSING (in module pybel.constants), 45
 FRAGMENT_START (in module pybel.constants), 44
 FRAGMENT_STOP (in module pybel.constants), 45
 from_bytes() (in module pybel), 24
 from_cbn_jgif() (in module pybel), 26
 from_cx() (in module pybel), 27
 from_cx_file() (in module pybel), 27
 from_cx_jsons() (in module pybel), 28
 from_database() (in module pybel), 29
 from_jgif() (in module pybel), 26
 from_json() (in module pybel), 25
 from_json_file() (in module pybel), 25
 from_jsons() (in module pybel), 25
 from_lines() (in module pybel), 22
 from_ndex() (in module pybel), 30
 from_path() (in module pybel), 22
 from_pickle() (in module pybel), 24
 from_url() (in module pybel), 23
 FUNCTION (in module pybel.constants), 40
 FUSION (in module pybel.constants), 41
 fusion (pybel.manager.models.Node attribute), 57
 fusion (pybel.parser.parse_bel.BelParser attribute), 50

G

GENE (in module pybel.constants), 41
 gene (pybel.parser.parse_bel.BelParser attribute), 51
 general_abundance (pybel.parser.parse_bel.BelParser attribute), 50
 get_annotations() (pybel.parser.parse_bel.BelParser method), 52
 get_annotations() (pybel.parser.parse_control.ControlParser method), 48
 get_bel_resource() (in module pybel.utils), 60
 get_cache_connection() (in module pybel.constants), 40
 get_edge_annotations() (pybel.BELGraph method), 12
 get_edge_citation() (pybel.BELGraph method), 12
 get_edge_evidence() (pybel.BELGraph method), 12
 get_node_description() (pybel.BELGraph method), 12
 get_node_label() (pybel.BELGraph method), 12
 get_node_name() (pybel.BELGraph method), 12

get_version() (in module pybel.utils), 60
 GMOD (in module pybel.constants), 41
 gmod (pybel.parser.parse_bel.BelParser attribute), 50
 gmod_namespace (in module pybel.parser.language), 46
 GMOD_ORDER (in module pybel.constants), 45
 GOCC_LATEST (in module pybel.constants), 40
 GRAPH_METADATA (in module pybel.constants), 44
 gsub (pybel.parser.parse_bel.BelParser attribute), 50
 GSUB_POSITION (in module pybel.constants), 45
 GSUB_REFERENCE (in module pybel.constants), 45
 GSUB_VARIANT (in module pybel.constants), 45

H

handle_aa_placeholder() (in module pybel.parser.language), 45
 handle_annotation_key() (pybel.parser.parse_control.ControlParser method), 48
 handle_annotation_list() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_annotation_owl() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_annotation_pattern() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_annotations_url() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_document() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_namespace_owl() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_namespace_pattern() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_namespace_url() (pybel.parser.parse_metadata.MetadataParser method), 47
 handle_term_singleton() (pybel.parser.parse_bel.BelParser method), 52
 has_annotation() (pybel.parser.parse_metadata.MetadataParser method), 48
 HAS_COMPONENT (in module pybel.constants), 42
 has_edge_citation() (pybel.BELGraph method), 12
 has_edge_evidence() (pybel.BELGraph method), 12
 has_enumerated_annotation() (pybel.parser.parse_metadata.MetadataParser method), 47

- has_enumerated_namespace() (pybel.parser.parse_identifier.IdentifierParser method), 49
 - has_enumerated_namespace() (pybel.parser.parse_metadata.MetadataParser method), 48
 - has_enumerated_namespace_name() (pybel.parser.parse_identifier.IdentifierParser method), 49
 - HAS_MEMBER (in module pybel.constants), 42
 - has_member_tag (pybel.parser.parse_bel.BelParser attribute), 52
 - has_namespace() (pybel.parser.parse_identifier.IdentifierParser method), 49
 - has_namespace() (pybel.parser.parse_metadata.MetadataParser method), 48
 - HAS_PRODUCT (in module pybel.constants), 42
 - HAS_REACTANT (in module pybel.constants), 42
 - has_regex_annotation() (pybel.parser.parse_metadata.MetadataParser method), 48
 - has_regex_namespace() (pybel.parser.parse_identifier.IdentifierParser method), 49
 - has_regex_namespace() (pybel.parser.parse_metadata.MetadataParser method), 48
 - has_regex_namespace_name() (pybel.parser.parse_identifier.IdentifierParser method), 49
 - has_singleton_terms (pybel.BELGraph attribute), 11
 - HAS_VARIANT (in module pybel.constants), 42
 - hash_edge() (in module pybel.utils), 62
 - hash_node() (in module pybel.utils), 61
 - HGVS (in module pybel.constants), 41
- I**
- ID (in module pybel.constants), 42
 - IDENTIFIER (in module pybel.constants), 41
 - IdentifierParser (class in pybel.parser.parse_identifier), 49
 - IllegalAnnotationValueWarning, 36
 - InconsistentDefinitionError, 34
 - INCREASES (in module pybel.constants), 43
 - increases_tag (pybel.parser.parse_bel.BelParser attribute), 51
 - InvalidCitationLengthException, 37
 - InvalidCitationType, 38
 - InvalidFunctionSemantic, 39
 - InvalidMetadataException, 36
 - InvalidPubMedIdentifierWarning, 38
 - INVERSE_DOCUMENT_KEYS (in module pybel.constants), 44
 - IS_A (in module pybel.constants), 43
 - is_a_tag (pybel.parser.parse_bel.BelParser attribute), 52
- is_int() (in module pybel.parser.utils), 62
 - is_url() (in module pybel.utils), 60
 - is_variant (pybel.manager.models.Node attribute), 57
- K**
- keyword (pybel.manager.models.Annotation attribute), 55
 - keyword (pybel.manager.models.Namespace attribute), 54
 - KIND (in module pybel.constants), 41
- L**
- LABEL (in module pybel.constants), 41
 - LexicographyWarning, 39
 - license (pybel.manager.models.Annotation attribute), 55
 - license (pybel.manager.models.Namespace attribute), 54
 - licenses (pybel.manager.models.Network attribute), 56
 - LINE (in module pybel.constants), 42
 - list2tuple() (in module pybel.utils), 60
 - location (pybel.parser.parse_bel.BelParser attribute), 50
- M**
- MalformedMetadataException, 36
 - MalformedTranslocationWarning, 39
 - manager (pybel.parser.parse_metadata.MetadataParser attribute), 46
 - METADATA_AUTHORS (in module pybel.constants), 44
 - METADATA_CONTACT (in module pybel.constants), 44
 - METADATA_COPYRIGHT (in module pybel.constants), 44
 - METADATA_DESCRIPTION (in module pybel.constants), 44
 - METADATA_DISCLAIMER (in module pybel.constants), 44
 - METADATA_INSERT_KEYS (in module pybel.constants), 44
 - METADATA_LICENSES (in module pybel.constants), 44
 - METADATA_NAME (in module pybel.constants), 44
 - METADATA_PROJECT (in module pybel.constants), 44
 - METADATA_VERSION (in module pybel.constants), 44
 - MetadataParser (class in pybel.parser.parse_metadata), 46
 - MIRNA (in module pybel.constants), 41
 - mirna (pybel.parser.parse_bel.BelParser attribute), 51
 - MissingAnnotationKeyWarning, 36
 - MissingAnnotationRegexWarning, 36
 - MissingBelResource, 36
 - MissingCitationException, 37
 - MissingDefaultNameWarning, 34
 - MissingMetadataException, 37
 - MissingNamespaceNameWarning, 35
 - MissingNamespaceRegexWarning, 35

- MissingSupportWarning, 38
- Modification (class in pybel.manager.models), 57
- modifier (pybel.manager.models.Property attribute), 59
- modName (pybel.manager.models.Modification attribute), 57
- modNamespace (pybel.manager.models.Modification attribute), 57
- modType (pybel.manager.models.Modification attribute), 57
- molecular_activity (pybel.parser.parse_bel.BelParser attribute), 51
- ## N
- NakedNameWarning, 34
- NAME (in module pybel.constants), 41
- name (pybel.BELGraph attribute), 11
- name (pybel.manager.models.Annotation attribute), 55
- name (pybel.manager.models.AnnotationEntry attribute), 56
- name (pybel.manager.models.Citation attribute), 58
- name (pybel.manager.models.Namespace attribute), 54
- name (pybel.manager.models.NamespaceEntry attribute), 54
- name (pybel.manager.models.Network attribute), 56
- Namespace (class in pybel.manager.models), 54
- NAMESPACE (in module pybel.constants), 41
- namespace_dict (pybel.parser.parse_identifier.IdentifierParser attribute), 49
- namespace_dict (pybel.parser.parse_metadata.MetadataParser attribute), 46
- NAMESPACE_DOMAIN_TYPES (in module pybel.constants), 40
- namespace_owl (pybel.BELGraph attribute), 11
- namespace_owl_dict (pybel.parser.parse_metadata.MetadataParser attribute), 47
- namespace_pattern (pybel.BELGraph attribute), 11
- namespace_regex (pybel.parser.parse_identifier.IdentifierParser attribute), 49
- namespace_regex (pybel.parser.parse_metadata.MetadataParser attribute), 47
- namespace_regex_compiled (pybel.parser.parse_identifier.IdentifierParser attribute), 49
- namespace_regex_compiled (pybel.parser.parse_metadata.MetadataParser attribute), 47
- namespace_url (pybel.BELGraph attribute), 11
- namespace_url_dict (pybel.parser.parse_metadata.MetadataParser attribute), 47
- NamespaceEntry (class in pybel.manager.models), 54
- NamespaceEntryEquivalence (class in pybel.manager.models), 55
- NamespaceIdentifierWarning, 34
- namespacePattern (pybel.manager.models.Node attribute), 57
- NameWarning, 34
- NEGATIVE_CORRELATION (in module pybel.constants), 43
- negative_correlation_tag (pybel.parser.parse_bel.BelParser attribute), 51
- nest() (in module pybel.parser.utils), 63
- NestedRelationWarning, 39
- Network (class in pybel.manager.models), 56
- Node (class in pybel.manager.models), 56
- ## O
- OBJECT (in module pybel.constants), 42
- one_of_tags() (in module pybel.parser.utils), 63
- ORTHOLOGOUS (in module pybel.constants), 43
- orthologous_tag (pybel.parser.parse_bel.BelParser attribute), 52
- ## P
- parse_bel_resource() (in module pybel.utils), 59
- parse_datetime() (in module pybel.utils), 61
- participant (pybel.manager.models.Property attribute), 59
- PARTNER_3P (in module pybel.constants), 41
- PARTNER_5P (in module pybel.constants), 41
- PATHOLOGY (in module pybel.constants), 42
- pathology (pybel.parser.parse_bel.BelParser attribute), 51
- PlaceholderAminoAcidWarning, 39
- PMOD (in module pybel.constants), 41
- pmod (pybel.parser.parse_bel.BelParser attribute), 50
- PMOD_CODE (in module pybel.constants), 45
- pmod_legacy_labels (in module pybel.parser.language), 46
- pmod_namespace (in module pybel.parser.language), 46
- PMOD_ORDER (in module pybel.constants), 45
- PMOD_POSITION (in module pybel.constants), 45
- position (pybel.manager.models.Modification attribute), 57
- POSITIVE_CORRELATION (in module pybel.constants), 43
- positive_correlation_tag (pybel.parser.parse_bel.BelParser attribute), 52
- prognostic_biomarker_tag (pybel.parser.parse_bel.BelParser attribute), 52
- PROGONSTIC_BIOMARKER_FOR (in module pybel.constants), 43
- Property (class in pybel.manager.models), 59
- propValue (pybel.manager.models.Property attribute), 59
- PROTEIN (in module pybel.constants), 41
- protein (pybel.parser.parse_bel.BelParser attribute), 51

psub (pybel.parser.parse_bel.BelParser attribute), 50
 PSUB_POSITION (in module pybel.constants), 45
 PSUB_REFERENCE (in module pybel.constants), 45
 PSUB_VARIANT (in module pybel.constants), 45
 pybel (module), 3
 pybel.constants (module), 39
 pybel.exceptions (module), 34
 pybel.io (module), 22
 pybel.io.cx (module), 27
 pybel.io.gpickle (module), 24
 pybel.io.jgif (module), 26
 pybel.io.ndex_utils (module), 30
 pybel.io.neo4j (module), 29
 pybel.io.nodelink (module), 25
 pybel.manager.database_io (module), 29
 pybel.manager.models (module), 53
 pybel.parser.language (module), 45
 pybel.parser.modifiers.fragment (module), 14
 pybel.parser.modifiers.fusion (module), 17
 pybel.parser.modifiers.gene_modification (module), 15
 pybel.parser.modifiers.gene_substitution (module), 13
 pybel.parser.modifiers.location (module), 20
 pybel.parser.modifiers.protein_modification (module), 16
 pybel.parser.modifiers.protein_substitution (module), 13
 pybel.parser.modifiers.truncation (module), 14
 pybel.parser.modifiers.variant (module), 12
 pybel.parser.parse_exceptions (module), 34
 pybel.parser.utils (module), 62
 pybel.struct (module), 10
 pybel.utils (module), 59
 PYBEL_CONNECTION (in module pybel.constants), 40
 PYBEL_DATA_DIR (in module pybel.constants), 40
 PYBEL_DIR (in module pybel.constants), 40
 PYBEL_EDGE_ALL_KEYS (in module pybel.constants), 42
 PYBEL_EDGE_DATA_KEYS (in module pybel.constants), 42
 PYBEL_LOG_DIR (in module pybel.constants), 40
 pybel_version (pybel.BELGraph attribute), 11
 PyBelParserWarning, 34
 PyBelWarning, 34

Q

query_url (pybel.manager.models.Namespace attribute), 54

R

RANGE_3P (in module pybel.constants), 41
 RANGE_5P (in module pybel.constants), 41
 rate_limit_tag (pybel.parser.parse_bel.BelParser attribute), 52
 RATE_LIMITING_STEP_OF (in module pybel.constants), 43
 reactants (pybel.parser.parse_bel.BelParser attribute), 51

REACTION (in module pybel.constants), 42
 RedefinedAnnotationError, 34
 RedefinedNamespaceError, 34
 reference (pybel.manager.models.Citation attribute), 58
 REGULATES (in module pybel.constants), 43
 regulates_tag (pybel.parser.parse_bel.BelParser attribute), 51
 RelabelWarning, 39
 RELATION (in module pybel.constants), 42
 relativeKey (pybel.manager.models.Property attribute), 59
 REQUIRED_METADATA (in module pybel.constants), 44
 RNA (in module pybel.constants), 41
 rna (pybel.parser.parse_bel.BelParser attribute), 51

S

session (pybel.manager.base_cache.BaseCacheManager attribute), 53
 sessionmaker (pybel.manager.base_cache.BaseCacheManager attribute), 53
 set_node_description() (pybel.BELGraph method), 12
 set_node_label() (pybel.BELGraph method), 12
 species (pybel.manager.models.Namespace attribute), 54
 subdict_matches() (in module pybel.utils), 62
 SUBJECT (in module pybel.constants), 42
 SUBPROCESS_OF (in module pybel.constants), 43
 subprocess_of_tag (pybel.parser.parse_bel.BelParser attribute), 52

T

text (pybel.manager.models.Evidence attribute), 58
 to_bel() (in module pybel), 23
 to_bel_lines() (in module pybel), 23
 to_bel_path() (in module pybel), 23
 to_bytes() (in module pybel), 24
 to_csv() (in module pybel), 28
 to_cx() (in module pybel), 27
 to_cx_file() (in module pybel), 27
 to_cx_jsons() (in module pybel), 28
 to_database() (in module pybel), 29
 to_graphml() (in module pybel), 28
 to_gsea() (in module pybel), 29
 to_jgif() (in module pybel), 26
 to_json() (in module pybel), 25
 to_json() (pybel.manager.models.AnnotationEntry method), 56
 to_json() (pybel.manager.models.Citation method), 58
 to_json() (pybel.manager.models.Evidence method), 58
 to_json() (pybel.manager.models.Modification method), 57
 to_json() (pybel.manager.models.NamespaceEntry method), 55
 to_json() (pybel.manager.models.Node method), 57

to_json() (pybel.manager.models.Property method), 59
 to_json_file() (in module pybel), 25
 to_jsons() (in module pybel), 26
 to_ndex() (in module pybel), 30
 to_neo4j() (in module pybel), 30
 to_pickle() (in module pybel), 24
 to_sif() (in module pybel), 28
 tokenize_version() (in module pybel.utils), 60
 transcribed_tag (pybel.parser.parse_bel.BelParser attribute), 52
 TRANSCRIBED_TO (in module pybel.constants), 42
 translated_tag (pybel.parser.parse_bel.BelParser attribute), 52
 TRANSLATED_TO (in module pybel.constants), 43
 translocation (pybel.parser.parse_bel.BelParser attribute), 51
 triple() (in module pybel.parser.utils), 63
 trunc (pybel.parser.parse_bel.BelParser attribute), 50
 TRUNCATION_POSITION (in module pybel.constants), 45
 TWO_WAY_RELATIONS (in module pybel.constants), 44
 type (pybel.manager.models.Annotation attribute), 55
 type (pybel.manager.models.Citation attribute), 58
 type (pybel.manager.models.Node attribute), 57

U

UndefinedAnnotationWarning, 36
 UndefinedNamespaceWarning, 35
 unqualified_edge_code (in module pybel.constants), 44
 unqualified_edges (in module pybel.constants), 44
 uploaded (pybel.manager.models.Annotation attribute), 55
 uploaded (pybel.manager.models.Namespace attribute), 54
 url (pybel.manager.models.Annotation attribute), 55
 url (pybel.manager.models.Namespace attribute), 54

V

valid_date() (in module pybel.utils), 61
 valid_date_version() (in module pybel.utils), 61
 variant (pybel.parser.parse_bel.BelParser attribute), 50
 version (pybel.BELGraph attribute), 11
 version (pybel.manager.models.Annotation attribute), 55
 version (pybel.manager.models.Namespace attribute), 54
 version (pybel.manager.models.Network attribute), 56
 VersionFormatWarning, 36

W

warnings (pybel.BELGraph attribute), 11