pronto

Release 2.4.1

Martin Larralde

Feb 19, 2021
**A Python frontend to ontologies.**

`pronto` is a Python agnostic library designed to work with ontologies. At the moment, it can parse OBO, OBO Graphs or OWL in RDF/XML format, ontologies on the local host or from an network location, and export ontologies to OBO or OBO Graphs (in JSON format).
Run `pip install pronto` in a shell to download the latest release and all its dependencies from PyPi, or have a look at the Installation page to find other ways to install `pronto`.

**Note:** `pronto` requires `fastobo`, an efficient and faultless parser for the OBO language implemented in Rust. Most platforms, such as Linux x86-64, OSX and Windows x86-64 provide precompiled packages, but other less frequent platforms will require a working Rust toolchain. See the `fastobo Installation page` and the Rust Forge tutorial for more information about this topic.
2.1 Installation

2.1.1 PyPi

`pronto` is hosted on GitHub, but the easiest way to install it is to download the latest release from its PyPi repository. It will install all dependencies then install the `pronto` module:

```
$ pip install --user pronto
```

2.1.2 Conda

Pronto is also available as a recipe in the bioconda channel. To install, simply use the `conda` installer:

```
$ conda install -c bioconda pronto
```

2.1.3 GitHub + pip

If, for any reason, you prefer to download the library from GitHub, you can clone the repository and install the repository by running (with the admin rights):

```
$ pip install --user https://github.com/althonos/pronto/archive/master.zip
```

Keep in mind this will install the development version of the library, so not everything may work as expected compared to a stable versioned release.

2.1.4 GitHub + setuptools

If you do not have `pip` installed, you can still clone the repository and run the `setup.py` file manually, although you will need to install the project dependencies yourself:

```
$ git clone https://github.com/althonos/pronto
$ cd pronto
# python setup.py install
```
2.2 Examples

2.2.1 Exploring MzML files with the MS Ontology

In this example, we will learn how to use pronto to extract a hierarchy from the MS Ontology, a controlled vocabulary developed by the Proteomics Standards Initiative to hold metadata about Mass Spectrometry instrumentation and other Protein Identification and Quantitation software. This example is taken from a real situation that kickstarted the development of pronto to extract metadata from MzML files, a file format for Mass Spectrometry data based on XML.

Loading ms.obo

The MS ontology is available online on the OBO Foundry, so unless we are using a local version we can simply use the version found online to load the OBO file. We may get some encoding warnings since ms.obo imports some legacy ontologies, but we should be OK for the most part since we are only querying terms directly.

```python
[1]: import pronto
ms = pronto.Ontology.from_obo_library("ms.obo")
```

Displaying a class hierarchy with Vega

The MS ontology contains a catalog of several instruments, grouped by instrument manufacturers, but not all instruments are at the same depth level. We can easily use the Term.subclasses method to find all instruments defined in the controlled vocabulary. Let’s then build a tree from all the subclasses of MS:1000031:

```python
[2]: instruments = ms['MS:1000031'].subclasses().to_set()
data = [
for term in instruments:
  value = {"id": int(term.id[3:]), "name": term.id, "desc": term.name}
  parents = term.superclasses(with_self=False, distance=1).to_set() & instruments
  if parents:
    value['parent'] = int(parents.pop().id[3:])
  data.append(value)
```

Now that we have our tree structure, we can render it simply with Vega to get a better idea of the classes we are inspecting:

```python
[3]: import json
import urllib.request

# Let’s use the Vega radial tree example as a basis of the visualization
view = json.load(urllib.request.urlopen("https://vega.github.io/vega/examples/radial-tree-layout.vg.json"))

# First replace the default data with our own
view['data'][0].pop('url')
view['data'][0]['values'] = data
```

(continues on next page)
view['marks'][1]['encode']['enter']['tooltip'] = {"signal": "datum.desc"}
view['signals'][4]['value'] = 'cluster'

# Render the clustered tree
display({'application/vnd.vega.v5+json': view}, raw=True)
Extracting the instruments from an MzML file

MzML files store the metadata corresponding to one or several MS scans using the MS controlled vocabulary, but the location and type of metadata can vary and needs to be extracted from a term subclassing hierarchy. Let’s download an example file from the MetaboLights library and parse it with xml.etree:

```python
import urllib.request
import xml.etree.ElementTree as etree
URL = "http://ftp.ebi.ac.uk/pub/databases/metabolights/studies/public/MTBLS341/pos_Exp2-K3_2-E,5_01_7458.d.mzML"
mzml = etree.parse(urllib.request.urlopen(URL))
```

Now we want to extract the instruments that were used in the MS scan, that are stored as mzml:cvParam elements: we build a set of all the instruments in the MS ontology, and we iterate over the instruments mzml:cvParam to find the ones that refer to instruments:

```python
instruments = ms["MS:1000031"].subclasses().to_set().ids
study_instruments = []
path = "mzml:instrumentConfigurationList/mzml:instrumentConfiguration/mzml:cvParam"
for element in mzml.iterfind(path, {'mzml': 'http://psi.hupo.org/ms/mzml'}):
    if element.attrib['accession'] in instruments:
        study_instruments.append(ms[element.attrib['accession']])
print(study_instruments)
[Term('MS:1000703', name='micrOTOF-Q')]
```

Finally we can extract the manufacturer of the instruments we found by checking which one of its superclasses is a direct child of the MS:1000031 term. We use the distance argument of subclasses to get the direct subclasses of instrument model, which are the manufacturers, and we use set operations to select manufacturers from the superclasses of each instrument we found.

```python
manufacturers = ms["MS:1000031"].subclasses(distance=1, with_self=False).to_set()
study_manufacturers = []
for instrument in study_instruments:
    study_manufacturers.extend(manufacturers & instrument.superclasses().to_set())
print(study_manufacturers)
[Term('MS:1000122', name='Bruker Daltonics instrument model')]
```

Validating the controlled vocabulary terms in an MzML file

All mzml:cvParam XML elements are required to have the 3 following attributes:

- `accession`, which is the identifier of the term in one of the ontologies imported in the file
- `cvRef`, which is the identifier of the ontology imported in the file
- `name`, which is the textual definition of the term

`name` in particular is redundant with respect to the actual ontology file, but can help rendering the XML elements. However, some MzML files can have a mismatch between the `name` and `accession` attributes. In order to check these mismatches we can use pronto to retrieve the name of all of these controlled vocabulary terms.
2.3 Updating from earlier versions

2.3.1 From v1.*

Update from v1.* to v2.* should be straightforward; the only reason the major version was updated was because the cache argument was removed from the Ontology constructor.

2.3.2 From v0.*

Render to OBO

Exporting an ontology to the OBO (or other supported formats) is now done with the dump and dumps methods:

```python
# before
print(ontology.obo)
open("out.obo", "w").write(ontology.obo)

# after
print(ontology.dumps(format="obo"))
ontology.dump(open("out.obo", "w"), format="obo")
```

Subclasses and superclasses

pronto is not opinionated about the direction of a relationship. Subclassing relationships are now handled as a special case, following the semantics of the rdfs:subClassOf property in the RDF schema.

Therefore, the code to access subclasses and superclasses of a Term has been updated:

```python
# before
children: pronto.TermList = term.rchildren()
parents: pronto.TermList = term.rparents()

# after
children_iter:Iterable[Term] = term.subclasses()
parents_iter:Iterable[Term] = term.superclasses()
```
Because we follow the RDF semantics, any class is also its own subclass and superclass; therefore, both of these iterators will yield the term itself as the first member of the iteration. This behaviour can be annoying, so you can disable it by giving `with_self=False` as an argument to only get *true* subclasses or superclasses:

```python
cpyonto, Release 2.4.1

children_iter: Iterable[Term] = term.subclasses(with_self=False)
parents_iter: Iterable[Term] = term.superclasses(with_self=False)
```

To only get the direct subclasses or superclasses (i.e., what `Term.children` and `Term.parents` used to do), pass `distance=1` as an argument as well:

```python
children: Iterable[Term] = term.subclasses(with_self=False, distance=1)
parents: Iterable[Term] = term.superclasses(with_self=False, distance=1)
```

Since querying of subclasses and superclasses now gives you an iterator, but your previous code was expecting a `TermList`, you can use the `to_set` method to obtain a `TermSet` which hopefully will prevent the rest of your code to require more update.

```python
# you can use `to_set` to get a `TermSet` from the iterator
children: pronto.TermSet = term.subclasses().to_set()
parents: pronto.TermSet = term.superclasses().to_set()
```

## 2.4 API Reference

A Python frontend to ontologies.

`pronto` is a Python agnostic library designed to work with ontologies. At the moment, it can parse ontologies in the OBO, OBO Graphs or OWL in RDF/XML format, on either the local host or from an network location, and export ontologies to OBO or OBO Graphs (in JSON format).

### Caution:
Only classes and modules reachable from the top-level package `pronto` are considered public and are guaranteed stable over Semantic Versioning. Use submodules (other than `warnings`) at your own risk!

### Note:
`pronto` implements proper type checking for most of the methods and properties exposed in the public classes. This reproduces the behaviour of the Python standard library, to avoid common errors. This feature does however increase overhead, but can be disabled by executing Python in optimized mode (with the `-O` flag). Parsing performances are not affected.

### See also:
Online documentation for this version of the library on [Read The Docs](https:// pronto.readthedocs.io/).
2.4.1 Ontology

An abstraction over a $SHOIN^{(D)}$ ontology.

```python
pronto.Ontology
```

An ontology storing terms and the relationships between them.

### Ontology

```python
class pronto.Ontology
```

An ontology storing terms and the relationships between them.

Ontologies can be loaded with `pronto` if they are serialized in any of the following ontology languages and formats at the moment:

- **Ontology Web Language 2** in RDF/XML format.
- **Open Biomedical Ontologies 1.4**.
- **OBO graphs** in JSON format.

#### metadata

A data structure storing the metadata about the current ontology, either extracted from the `owl:Ontology` XML element or from the header of the OBO file.

Type: `Metadata`

#### timeout

The timeout in seconds to use when performing network I/O, for instance when connecting to the OBO library to download imports. This is kept for reference, as it is not used after the initialization of the ontology.

Type: `int`

#### imports

A dictionary mapping references found in the import section of the metadata to resolved `Ontology` instances.

Type: `Dict[str, Ontology]`

```python
classmethod from_obo_library(
    slug: str,
    import_depth: int = -1,
    timeout: int = 5,
    threads: Optional[int] = None
) → pronto.ontology.Ontology
```

Create an `Ontology` from a file in the OBO Library.

This is basically just a shortcut constructor to avoid typing the full OBO Library URL each time.

**Parameters**

- **slug (str)** – The filename of the ontology release to download from the OBO Library, including the file extension (should be one of `.obo`, `.owl` or `.json`).
- **import_depth (int)** – The maximum depth of imports to resolve in the ontology tree. Note that the library may not behave correctly when not importing the complete dependency tree, so you should probably use the default value and import everything.
- **timeout (int)** – The timeout in seconds to use when performing network I/O, for instance when connecting to the OBO library to download imports.
- **threads (int)** – The number of threads to use when parsing, for parsers that support multithreading. Give `None` to autodetect the number of CPUs on the host machine.
Example

```python
>>> apo = pronto.Ontology.from_obo_library("apo.obo")
>>> apo.metadata.ontology
'apo'
>>> apo.path
'http://purl.obolibrary.org/obo/apo.obo'
```

__init__

```python
__init__(handle: Union[BinaryIO, str, os.PathLike, None] = None, import_depth: int = -1, timeout: int = 5, threads: Optional[int] = None)
```

Create a new `Ontology` instance.

**Parameters**

- **handle** *(str, BinaryIO, PathLike, or None)* – Either the path to a file or a
  binary file handle that contains a serialized version of the ontology. If `None` is given, an
  empty `Ontology` is returned and can be populated manually.

- **import_depth** *(int)* – The maximum depth of imports to resolve in the ontology
  tree. *Note that the library may not behave correctly when not importing the complete
  dependency tree, so you should probably use the default value and import everything.*

- **timeout** *(int)* – The timeout in seconds to use when performing network I/O, for in-
  stance when connecting to the OBO library to download imports.

- **threads** *(int)* – The number of threads to use when parsing, for parsers that support
  multithreading. Give `None` to autodetect the number of CPUs on the host machine.

**Raises**

- **TypeError** – When the given `handle` could not be used to parse and ontology.

- **ValueError** – When the given `handle` contains a serialized ontology not supported
  by any of the builtin parsers.

__len__() → int

Return the number of entities in the ontology.

This method takes into accounts the terms and the relationships defined in the current ontology as well as all of its imports. To only count terms or relationships, use `len` on the iterator returned by the dedicated methods (e.g. `len(ontology.terms())`).

Example

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> len(ms)
6023
>>> len(ms.terms())
5995
```

__iter__() → pronto.utils.iter.SizedIterator[str][str]

Yield the identifiers of all the entities part of the ontology.

__getitem__(id: str) → Union[pronto.term.Term, pronto.relationship.Relationship]

Get any entity in the ontology graph with the given identifier.

__repr__()

Return a textual representation of `self` that should roundtrip.
__eq__(other)  
Return self==value.

__ge__()  
Return self>=value.

__gt__()  
Return self>value.

__le__()  
Return self<=value.

__lt__()  
Return self<value.

get(k, d) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D’s items

keys() → a set-like object providing a view on D’s keys

values() → an object providing a view on D’s values

dump(file: BinaryIO, format: str = ‘obo’)  
Serialize the ontology to a given file-handle.

Parameters

• file (BinaryIO) – A binary file handle open in reading mode to write the serialized ontology into.

• format (str) – The serialization format to use. Currently supported formats are: obo, json.

Example

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> with open("ms.json", "wb") as f:
...    ms.dump(f, format="json")
```

dumps(format: str = ‘obo’) → str  
Get a textual representation of the serialization ontology.

Example

```python
>>> go = pronto.Ontology("go.obo")
>>> print(go.dumps())
format-version: 1.2
data-version: releases/2019-07-01
...
```

synonym_types() → pronto.utils.iter.SizedIterator[pronto.synonym.SynonymType][pronto.synonym.SynonymType]  
Iterate over the synonym types of the ontology graph.

terms() → pronto.ontology._OntologyTerms  
Query the terms of an ontology.

Example
>>> pato = pronto.Ontology.from_obo_library("pato.obo")
>>> len(pato.terms())
2661
>>> "PATO:0000186" in pato.terms()
True
>>> for term in sorted(pato.terms()):
...    print(term)
Term('PATO:0000000', name='obsolete pato')
Term('PATO:0000001', name='quality')
...

relationships() → pronto.ontology._OntologyRelationships
Query the relationships of an ontology.

Example

>>> pato = pronto.Ontology.from_obo_library("pato.obo")
>>> len(pato.relationships())
24
>>> "reciprocal_of" in pato.relationships()
True
>>> for relationship in pato.relationships():
...    print(relationship)
  Relationship('correlates_with', ...)
  Relationship('decreased_in_magnitude_relative_to', ...)
...

create_term(id: str) → pronto.term.Term
Create a new term with the given identifier.

  Returns  Term – the newly created term view, which attributes can the be modified directly.

  Raises ValueError – if the provided id already identifies an entity in the ontology graph, or
  if it is not a valid OBO identifier.

create_relationship(id: str) → pronto.relationship.Relationship
Create a new relationship with the given identifier.

  Raises ValueError – if the provided id already identifies an entity in the ontology graph.

get_term(id: str) → pronto.term.Term
Get a term in the ontology graph from the given identifier.

  Raises KeyError – if the provided id cannot be found in the terms of the ontology graph.

get_relationship(id: str) → pronto.relationship.Relationship
Get a relationship in the ontology graph from the given identifier.

    Builtin ontologies (is_a and has_subclass) can be accessed with this method.

  Raises KeyError – if the provided id cannot be found in the relationships of the ontology
  graph.

get_synonym_type(id: str) → pronto.synonym.SynonymType
Get a synonym type in the ontology graph from the given identifier.

  Raises KeyError – if the provided id does not resolve to a synonym type declared in this
  Ontology or one of its imports.
2.4.2 View Layer

The following classes are part of the view layer, and store references to the ontology/entity they were declared in for verification purposes. For instance, this lets pronto check that a Synonym type can only be changed for a type declared in the Ontology header.

Because of this reason, none of these classes should be created manually, but obtained from methods of existing Ontology or Entity instances, such as Ontology.get_term to get a new Term.

<table>
<thead>
<tr>
<th>pronto.Entity</th>
<th>An entity in the ontology graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pronto.Relationship</td>
<td>A relationship, constitute the edges of the ontology graph.</td>
</tr>
<tr>
<td>pronto.Synonym</td>
<td>A synonym for an entity, with respect to the OBO terminology.</td>
</tr>
<tr>
<td>pronto.Term</td>
<td>A term, corresponding to a node in the ontology graph.</td>
</tr>
<tr>
<td>pronto.TermSet</td>
<td>A specialized mutable set to store Term instances.</td>
</tr>
</tbody>
</table>

**Entity**

class pronto.Entity

An entity in the ontology graph.

With respects to the OBO semantics, an Entity is either a term or a relationship in the ontology graph. Any entity has a unique identifier as well as some common properties.

__eq__(other: Any) → bool
Return self==value.
__lt__(other)
Return self<value.
__le__(other)
Return self<=value.
__gt__(other)
Return self>value.
__ge__(other)
Return self>=value.
__hash__()
Return hash(self).
__repr__()
Return repr(self).

property alternate_ids
A set of alternate IDs for this entity.

Type set of str

property annotations
Annotations relevant to the entity.

Type set of PropertyValue

property anonymous
Whether or not the entity has an anonymous id.

Semantics of anonymous entities are the same as B-Nodes in RDF.
property `builtin`

Whether or not the entity is built-in to the OBO format.

`pronto` uses this tag on the `is_a` relationship, which is the axiomatic to the OBO language but treated as a relationship in the library.

**Type** `bool`

property `comment`

A comment about the current entity.

Comments in `comment` clauses are guaranteed to be conserved by OBO parsers and serializers, unlike bang comments. A non `None` comment is semantically equivalent to an `rdfs:comment` in OWL2. When parsing from OWL, several RDF comments will be merged together into a single `comment` clause spanning over multiple lines.

**Type** `str` or `None`

property `consider`

A set of potential substitutes for an obsolete term.

An obsolete entity can provide one or more entities which may be appropriate substitutes, but needs to be looked at carefully by a human expert before the replacement is done.

**See also:**

`replaced_by`, which provides a set of entities suitable for automatic replacement.

**Type** `EntitySet`

property `created_by`

The name of the creator of the entity, if any.

This property gets translated to a `dc:creator` annotation in OWL2, which has very broad semantics. Some OBO ontologies may instead use other annotation properties such as the ones found in Information Interchange Ontology, which can be accessed in the `annotations` attribute of the entity, if any.

**Type** `str` or `None`

property `creation_date`

The date the entity was created.

**Type** `datetime` or `None`

property `definition`

The definition of the current entity.

Definitions in OBO are intended to be human-readable text describing the entity, with some additional cross-references if possible.

**Example**

```python
>>> hp = pronto.Ontology.from_obo_library("hp.obo")
>>> term = hp["HP:0009882"]
>>> term.name
'Short distal phalanx of finger'
>>> str(term.definition)
'Short distance from the end of the finger to the most distal...

>>> sorted(term.definition.xrefs)
[Xref('HPO:probinson'), Xref('PMID:19125433')]
```
Type Definition or None

property disjoint_from
The entities declared as disjoint from this entity.

Two entities are disjoint if they have no instances in common. Two entities that are disjoint cannot share any subentities, but the opposite is not always true.

Type EntitySet

property equivalent_to
The entities declared as equivalent to this entity.

Type EntitySet

property id
The OBO identifier of the entity.

Identifiers can be either prefixed (e.g. MS:1000031), unprefixed (e.g. part_of) or given as plain URLs. Identifiers cannot be edited.

Type str

property name
The name of the entity.

Names are formally equivalent to rdf:label in OWL2. The OBO format version 1.4 made names optional to improve OWL interoperability, as labels are optional in OWL.

Type str or None

property namespace
The namespace this entity is defined in.

Type str or None

property obsolete
Whether or not the entity is obsolete.

Hint: All OBO entities can be made obsolete through a boolean flag, and map to one or several replacements. When querying an obsolete entity, pronto will not attempt to perform any kind of replacement itself

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> term = ms["MS:1001414"]
>>> term
Term('MS:1001414', name='MGF scans')
>>> term.obsolete
True
```

To always get the up-to-date, non-obsolete entity, you could use the following snippet, going through a term replacement if there is no ambiguity

```python
>>> while term.obsolete:
...     if len(term.replaced_by) != 1:
...         raise ValueError(f"no replacement for {term.id}")
...     term = term.replaced_by.pop()
>>> term
Term('MS:1000797', name='peak list scans')
```
See also:

`consider` and `replaced_by`, storing some replacement options for an obsolete entity.

**Type** bool

**property replaced_by**
A set of replacements for an obsolete entity.

An obsolete entity can provide one or more replacement that can safely be used to automatically reassign instances to non-obsolete classes.

See also:

`consider`, which provides a set of entities suitable for replacement but requiring expert curation.

**Type** EntitySet

**property subsets**
The subsets containing this entity.

**Type** frozenset of str

**property synonyms**
A set of synonyms for this entity.

**Type** frozenset of Synonym

**property xrefs**
A set of database cross-references.

Xrefs can be used to describe an analogous entity in another vocabulary, such as a database or a semantic knowledge base.

**Type** frozenset of Xref

Add a new synonym to the current entity.

**Parameters**

- **description** (str) – The alternate definition of the entity, or a related human-readable synonym.

- **scope** (str or None) – An optional synonym scope. Must be either EXACT, RELATED, BROAD or NARROW if given.

- **type** (SynonymType or None) – An optional synonym type. Must be declared in the header of the current ontology.

- **xrefs** (iterable of Xref, or None) – A collections of database cross-references backing the origin of the synonym.

**Raises** ValueError – when given an invalid synonym type or scope.

**Returns** Synonym – A new synonym for the terms. The synonym is already added to the Entity.synonyms collection.
Relationship

class pronto.Relationship
   A relationship, constitute the edges of the ontology graph.
   Also sometimes referred as typedefs, relationship types, properties or predicates. Formally equivalent to a property (either ObjectProperty or AnnotationProperty) in OWL2.

   __eq__(other: Any) → bool
      Return self==value.

   __ge__(other)
      Return self>=value.

   __gt__(other)
      Return self>value.

   __hash__()
      Return hash(self).

   __le__(other)
      Return self<=value.

   __lt__(other)
      Return self<value.

   __repr__()
      Return repr(self).

      Add a new synonym to the current entity.

      Parameters
         • description(str) – The alternate definition of the entity, or a related human-readable synonym.
         • scope (str or None) – An optional synonym scope. Must be either EXACT, RELATED, BROAD or NARROW if given.
         • type (SynonymType or None) – An optional synonym type. Must be declared in the header of the current ontology.
         • xrefs (iterable of Xref, or None) – A collections of database cross-references backing the origin of the synonym.

      Raises ValueError – when given an invalid synonym type or scope.

      Returns Synonym – A new synonym for the terms. The synonym is already added to the Entity.synonyms collection.

   property alternate_ids
      A set of alternate IDs for this entity.

      Type set of str

   property annotations
      Annotations relevant to the entity.

      Type set of PropertyValue

2.4. API Reference
**property anonymous**

Whether or not the entity has an anonymous id.

Semantics of anonymous entities are the same as B-Nodes in RDF.

Type bool

**property builtin**

Whether or not the entity is built-in to the OBO format.

pronto uses this tag on the is_a relationship, which is the axiomatic to the OBO language but treated as a relationship in the library.

Type bool

**property comment**

A comment about the current entity.

Comments in comment clauses are guaranteed to be conserved by OBO parsers and serializers, unlike bang comments. A non None comment is semantically equivalent to a rdfs:comment in OWL2. When parsing from OWL, several RDF comments will be merged together into a single comment clause spanning over multiple lines.

Type str or None

**property consider**

A set of potential substitutes for an obsolete term.

An obsolete entity can provide one or more entities which may be appropriate substitutes, but needs to be looked at carefully by a human expert before the replacement is done.

See also:

replaced_by, which provides a set of entities suitable for automatic replacement.

Type EntitySet

**property created_by**

The name of the creator of the entity, if any.

This property gets translated to a dc:creator annotation in OWL2, which has very broad semantics. Some OBO ontologies may instead use other annotation properties such as the ones found in Information Interchange Ontology, which can be accessed in the annotations attribute of the entity, if any.

Type str or None

**property creation_date**

The date the entity was created.

Type datetime or None

**property definition**

The definition of the current entity.

Definitions in OBO are intended to be human-readable text describing the entity, with some additional cross-references if possible.

---

**Example**

```python
>>> hp = pronto.Ontology.from_obo_library("hp.obo")
>>> term = hp["HP:0009882"]
>>> term.name
```
'Short distal phalanx of finger'
>>> str(term.definition)
'Short distance from the end of the finger to the most distal...

>>> sorted(term.definition.xrefs)
[Xref('HPO:probinson'), Xref('PMID:19125433')]

Type **Definition** or None

**property disjoint_from**
The entities declared as disjoint from this entity.

Two entities are disjoint if they have no instances in common. Two entities that are disjoint cannot share any subentities, but the opposite is not always true.

Type **EntitySet**

**property equivalent_to**
The entities declared as equivalent to this entity.

Type **EntitySet**

**property id**
The OBO identifier of the entity.

Identifiers can be either prefixed (e.g. MS:1000031), unprefixed (e.g. part_of) or given as plain URLs. Identifiers cannot be edited.

Type **str**

**property name**
The name of the entity.

Names are formally equivalent to rdf:label in OWL2. The OBO format version 1.4 made names optional to improve OWL interoperability, as labels are optional in OWL.

Type **str** or None

**property namespace**
The namespace this entity is defined in.

Type **str** or None

**property obsolete**
Whether or not the entity is obsolete.

**Hint**: All OBO entities can be made obsolete through a boolean flag, and map to one or several replacements. When querying an obsolete entity, **pronto** will not attempt to perform any kind of replacement itself.

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> term = ms["MS:1001414"]
>>> term
Term('MS:1001414', name='MGF scans')
>>> term.obsolete
True
```

To always get the up-to-date, non-obsolete entity, you could use the following snippet, going through a term replacement if there is no ambiguity.
>>> while termobsolete:
...    if len(term.replaced_by) != 1:
...        raise ValueError(f"no replacement for {term.id}")
...    term = term.replaced_by.pop()
>>> term
Term('MS:1000797', name='peak list scans')

See also:

consider and replaced_by, storing some replacement options for an obsolete entity.

Type bool

property replaced_by
A set of replacements for an obsolete term.

An obsolete entity can provide one or more replacements that can safely be used to automatically reassign instances to non-obsolete classes.

See also:

consider, which provides a set of entities suitable for replacement but requiring expert curation.

Type EntitySet

property subsets
The subsets containing this entity.

Type frozenset of str

property synonyms
A set of synonyms for this entity.

Type frozenset of Synonym

property xrefs
A set of database cross-references.

Xrefs can be used to describe an analogous entity in another vocabulary, such as a database or a semantic knowledge base.

Type frozenset of Xref

subproperties (distance: Optional[int] = None, with_self: bool = True) -> pronto.logic.lineage.SubpropertiesHandler
Get an handle over the subproperties of this Relationship.

Parameters

- distance (int, optional) — The maximum distance between this relationship and the yielded subproperties (0 for the relationship itself, 1 for its immediate children, etc.). Use None to explore the entire directed graph transitively.

- with_self (bool) — Whether or not to include the current term in the terms being yielded. RDF semantics state that the rdfs:subClassOf property is reflexive (and therefore is rdfs:subPropertyOf reflexive too by transitivity), so this is enabled by default, but in most practical cases only the distinct subproperties are desired.

superproperties (distance: Optional[int] = None, with_self: bool = True) -> pronto.logic.lineage.SuperpropertiesHandler
Get an handle over the superproperties of this Relationship.
In order to follow the semantics of \texttt{rdf:subPropertyOf}, which in turn respects the mathematical definition of subset inclusion, \texttt{is_a} is defined as a transitive relationship, hence the inverse relationship is also transitive by closure property.

**Parameters**

- \texttt{distance (int, optional)} – The maximum distance between this relationship and the yielded subproperties (0 for the relationship itself, 1 for its immediate parents, etc.). Use \texttt{None} to explore the entire directed graph transitively.
- \texttt{with_self (bool)} – Whether or not to include the current term in the terms being yielded. RDF semantics state that the \texttt{rdfs:subClassOf} property is transitive (and therefore is \texttt{rdfs:subPropertyOf} transitive too), so this is enabled by default, but in most practical cases only the distinct subproperties are desired.

**property antisymmetric**

Whether this relationship is anti-symmetric.

Type \texttt{bool}

**property asymmetric**

Whether this relationship is asymmetric.

Type \texttt{bool}

**property class_level**

Whether this relationship is applied at class level.

This tag affects how OBO relationship tags should be translated in OWL2: by default, all relationship tags are taken to mean an all-some relation over an instance level relation. With this flag set to \texttt{True}, the relationship will be translated to an \texttt{owl:hasValue} restriction.

Type \texttt{bool}

**property cyclic**

Whether this relationship is cyclic.

Type \texttt{bool}

**property disjoint_over**

The relationships this relationships is disjoint over.

Type \texttt{frozenset}

**property domain**

The domain of the relationship, if any.

Type \texttt{Term} or \texttt{None}

**property functional**

Whether this relationship is functional.

Type \texttt{bool}

**property inverse_functional**

Whether this relationship is inverse functional.

Type \texttt{bool}

**property metadata_tag**

Whether or not this relationship is a metadata tag.

This tag affects how OBO typedefs should be translated in OWL2: by default, all typedef tags are translated to an \texttt{owl:ObjectProperty}. With this flag set to \texttt{True}, the typedef will be translated to an \texttt{owl:AnnotationProperty}. 
Type `bool`

**property holds_over_chain**

The chains this relationship holds over.

Type `frozenset of Relationship` couples

**property inverse_of**

The inverse of this relationship, if any.

Type `Relationship` or `None`

**property intersection_of**

The relations this relationship is an intersection of.

Type `RelationshipSet`

**property range**

The range of the relationship, if any.

Type `Term` or `None`

**property reflexive**

Whether or not the relationship is reflexive.

Type `bool`

**property symmetric**

Whether or not the relationship is symmetric.

Type `bool`

**property transitive**

Whether or not the relationship is transitive.

Type `bool`

**property transitive_over**

The relations this relationship is transitive over.

Type `RelationshipSet`

**Synonym**

class `pronto.Synonym`

A synonym for an entity, with respect to the OBO terminology.

```python
__ge__(other, NotImplemented=NotImplemented)
    Return a >= b. Computed by @total_ordering from (not a < b).
__gt__(other, NotImplemented=NotImplemented)
    Return a > b. Computed by @total_ordering from (not a < b) and (a != b).
__le__(other, NotImplemented=NotImplemented)
    Return a <= b. Computed by @total_ordering from (a < b) or (a == b).
__eq__(other: object)
    Return self==value.
__lt__(other: object)
    Return self<value.
__hash__()
    Return hash(self).
```
__repr__()  
Return repr(self).

Term

class pronto.Term
A term, corresponding to a node in the ontology graph.

Formally a Term frame is equivalent to an owl:Class declaration in OWL2 language. However, some constructs may not be possible to express in both OBO and OWL2.

Term should not be manually instantiated, but obtained from an existing Ontology instance, using either the create_term or the get_term method.

__eq__(other: Any) → bool  
Return self==value.

__ge__(other)  
Return self>=value.

__gt__(other)  
Return self>value.

__hash__()  
Return hash(self).

__le__(other)  
Return self<=value.

__lt__(other)  
Return self<value.

__repr__()  
Return repr(self).

Add a new synonym to the current entity.

Parameters

• description (str) – The alternate definition of the entity, or a related human-readable synonym.

• scope (str or None) – An optional synonym scope. Must be either EXACT, RELATED, BROAD or NARROW if given.

• type (SynonymType or None) – An optional synonym type. Must be declared in the header of the current ontology.

• xrefs (iterable of Xref, or None) – A collections of database cross-references backing the origin of the synonym.

Raises ValueError – when given an invalid synonym type or scope.

Returns Synonym – A new synonym for the terms. The synonym is already added to the Entity.synonyms collection.

property alternate_ids
A set of alternate IDs for this entity.

Type set of str

2.4. API Reference 25
property annotations
  Annotations relevant to the entity.
  Type set of PropertyValue

property anonymous
  Whether or not the entity has an anonymous id.
  Semantics of anonymous entities are the same as B-Nodes in RDF.
  Type bool

property builtin
  Whether or not the entity is built-in to the OBO format.
  pronto uses this tag on the is_a relationship, which is the axiomatic to the OBO language but treated as a relationship in the library.
  Type bool

property comment
  A comment about the current entity.
  Comments in comment clauses are guaranteed to be conserved by OBO parsers and serializers, unlike bang comments. A non None comment is semantically equivalent to a rdfs:comment in OWL2. When parsing from OWL, several RDF comments will be merged together into a single comment clause spanning over multiple lines.
  Type str or None

property consider
  A set of potential substitutes for an obsolete term.
  An obsolete entity can provide one or more entities which may be appropriate substitutes, but needs to be looked at carefully by a human expert before the replacement is done.
  See also:
  replaced_by, which provides a set of entities suitable for automatic replacement.
  Type EntitySet

property created_by
  The name of the creator of the entity, if any.
  This property gets translated to a dc:creator annotation in OWL2, which has very broad semantics. Some OBO ontologies may instead use other annotation properties such as the ones found in Information Interchange Ontology, which can be accessed in the annotations attribute of the entity, if any.
  Type str or None

property creation_date
  The date the entity was created.
  Type datetime or None

property definition
  The definition of the current entity.
  Definitions in OBO are intended to be human-readable text describing the entity, with some additional cross-references if possible.

Example
>>> hp = pronto.Ontology.from_obo_library("hp.obo")
>>> term = hp["HP:0009882"]
>>> term.name
'Short distal phalanx of finger'
>>> str(term.definition)
'Short distance from the end of the finger to the most distal...

>>> sorted(term.definition.xrefs)
[Xref('HPO:probinson'), Xref('PMID:19125433')]

Type  Definition or None

property disjoint_from
The entities declared as disjoint from this entity.
Two entities are disjoint if they have no instances in common. Two entities that are disjoint cannot share any subentities, but the opposite is not always true.

Type  EntitySet

property equivalent_to
The entities declared as equivalent to this entity.

Type  EntitySet

property id
The OBO identifier of the entity.
Identifiers can be either prefixed (e.g. MS:1000031), unprefixed (e.g. part_of) or given as plain URLs. Identifiers cannot be edited.

Type  str

property name
The name of the entity.
Names are formally equivalent to rdf:label in OWL2. The OBO format version 1.4 made names optional to improve OWL interoperability, as labels are optional in OWL.

Type  str or None

property namespace
The namespace this entity is defined in.

Type  str or None

property obsolete
Whether or not the entity is obsolete.

Hint: All OBO entities can be made obsolete through a boolean flag, and map to one or several replacements. When querying an obsolete entity, pronto will not attempt to perform any kind of replacement itself.

>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> term = ms["MS:1001414"]
>>> term
Term('MS:1001414', name='MGF scans')
>>> termobsolete
True
To always get the up-to-date, non-obsolete entity, you could use the following snippet, going through a term replacement if there is no ambiguity

```python
>>> while term.obsolete:
...     if len(term.replaced_by) != 1:
...         raise ValueError(f"no replacement for {term.id}"")
...     term = term.replaced_by.pop()

>>> term
Term('MS:1000797', name='peak list scans')
```

See also:

`consider` and `replaced_by`, storing some replacement options for an obsolete entity.

**property replaced_by**

A set of replacements for an obsolete term.

An obsolete entity can provide one or more replacement that can safely be used to automatically reassign instances to non-obsolete classes.

See also:

`consider`, which provides a set of entities suitable for replacement but requiring expert curation.

**property subsets**

The subsets containing this entity.

**property synonyms**

A set of synonyms for this entity.

**property xrefs**

A set of database cross-references.

Xrefs can be used to describe an analogous entity in another vocabulary, such as a database or a semantic knowledge base.

**objects**

Iterate over the terms `t` verifying `self \cdot r \cdot t`.

**Example**

```python
>>> go = pronto.Ontology.from_obo_library("go.obo")
>>> go['GO:0048870']
Term('GO:0048870', name='cell motility')
>>> list(go['GO:0048870'].objects(go.get_relationship('part_of')))
[Term('GO:0051674', name='localization of cell')]
```
Todo: Make `Term.objects` take in account `holds_over_chain` and `transitive_over` values of the relationship it is building an iterator with.

```python
superclasses(distance: Optional[int] = None, with_self: bool = True) →
pronto.logic.lineage.SuperclassesHandler
Get an handle over the superclasses of this `Term`.
```

In order to follow the semantics of `rdf:subClassOf`, which in turn respects the mathematical definition of subset inclusion, `is_a` is defined as a reflexive relationship, and so is its inverse relationship.

**Parameters**

- **distance** (*int, optional*) – The maximum distance between this term and the yielded superclass (0 for the term itself, 1 for its immediate superclasses, etc.). Use `None` to explore transitively the entire directed graph.

- **with_self** (*bool*) – Whether or not to include the current term in the terms being yielded. RDF semantics state that the `rdfs:subClassOf` property is reflexive, so this is enabled by default, but in most practical cases only the distinct subclasses are desired.

**Yields** `Term` – Superclasses of the selected term, breadth-first. The first element is always the term itself, use `itertools.islice` to skip it.

**Example**

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> sup = iter(ms['MS:1000143'].superclasses())
>>> next(sup)
Term('MS:1000143', name='API 150EX')
>>> next(sup)
Term('MS:1000121', name='SCIEX instrument model')
>>> next(sup)
Term('MS:1000031', name='instrument model')
```

**Note:** The time complexity for this algorithm is in $O(n)$, where $n$ is the number of subclasses of initial term.

**See also:**

The RDF Schema 1.1 specification, defining the `rdfs:subClassOf` property, which the `is_a` relationship is translated to in OWL2 language.

```python
subclasses(distance: Optional[int] = None, with_self: bool = True) →
pronto.logic.lineage.SubclassesHandler
Get an handle over the subclasses of this `Term`.
```

**Parameters**

- **distance** (*int, optional*) – The maximum distance between this term and the yielded subclass (0 for the term itself, 1 for its immediate children, etc.). Use `None` to explore the entire directed graph transitively.

- **with_self** (*bool*) – Whether or not to include the current term in the terms being yielded. RDF semantics state that the `rdfs:subClassOf` property is reflexive, so this
is enabled by default, but in most practical cases only the distinct subclasses are desired.

Yields Term – Subclasses of the selected term, breadth-first. The first element is always the term itself, use `itertools.islice` to skip it.

### Example

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> sub = iter(ms['MS:1000031'].subclasses())
>>> next(sub)
Term('MS:1000031', name='instrument model')
>>> next(sub)
Term('MS:1000121', name='SCIEX instrument model')
>>> next(sub)
Term('MS:1000122', name='Bruker Daltonics instrument model')
```

**Hint:** Use the `to_set` method of the returned iterator to efficiently collect all subclasses into a `TermSet`.

**Note:** This method has a runtime that is $O(n)$ where $n$ is the number of subclasses of the initial term. While OBO and OWL only explicit the `superclassing` relationship (equivalent to the `rdfs:subClassOf` property in RDF), we can build a cache that stores the edges of the resulting knowledge graph in an index accessible by both endpoints of each edge.

`is_leaf()` → bool

Check whether the term is a leaf in the ontology.

We define leaves as nodes in the ontology which do not have subclasses since the subclassing relationship is directed and can be used to create a DAG of all the terms in the ontology.

### Example

```python
>>> ms = pronto.Ontology.from_obo_library("ms.obo")
>>> ms['MS:1000031'].is_leaf() # instrument model
False
>>> ms['MS:1001792'].is_leaf() # Xevo TQ-S
True
```

**Note:** This method has a runtime of $O(1)$ as `Ontology` objects internally cache the subclasses of each term.

**property intersection_of**

The terms this term is an intersection of.

Type `frozenset`
**TermSet**

```python
class pronto.TermSet
    A specialized mutable set to store Term instances.

    __eq__(other)
    Return self==value.

    __ge__(other)
    Return self>=value.

    __gt__(other)
    Return self>value.

    __le__(other)
    Return self<=value.

    __lt__(other)
    Return self<value.

    __repr__()
    Return repr(self).

classmethod _from_iterable(it)
    Construct an instance of the class from any iterable input.
    Must override this method if the class constructor signature does not accept an iterable for an input.

    _hash()
    Compute the hash value of a set.
    Note that we don’t define __hash__: not all sets are hashable. But if you define a hashable set type, its
    __hash__ should call this function.
    This must be compatible __eq__.
    All sets ought to compare equal if they contain the same elements, regardless of how they are implemented,
    and regardless of the order of the elements; so there’s not much freedom for __eq__ or __hash___. We match
    the algorithm used by the built-in frozenset type.

    add(entity: _E) → None
    Add an element.

    clear() → None
    This is slow (creates N new iterators!) but effective.

    discard(entity: _E) → None
    Remove an element. Do not raise an exception if absent.

    isdisjoint(other)
    Return True if two sets have a null intersection.

    pop() → _E
    Return the popped value. Raise KeyError if empty.

    remove(entity: _E)
    Remove an element. If not a member, raise a KeyError.

    subclasses(distance: Optional[int] = None, with_self: bool = True) →
    pronto.logic.lineage.SubclassesIterator
    Get an iterator over the subclasses of all terms in the set.
```

2.4. API Reference
Caution: Contrary to `pronto.Term.subclasses`, this method does not return a handler that lets you edit the subclasses directly. Adding a new subclass to all the members of the set must be done explicitly.

```python
superclasses(distance: Optional[int] = None, with_self: bool = True) \rightarrow 
pronto.logic.lineage.SuperclassesIterator
```

Get an iterator over the superclasses of all terms in the set.

Caution: Contrary to `pronto.Term.superclasses`, this method does not return a handler that lets you edit the superclasses directly. Adding a new superclass to all the members of the set must be done explicitly.

Example

```python
>>> ms = pronto.Ontology("ms.obo")
>>> s = pronto.TermSet({ms['MS:1000122'], ms['MS:1000124']})
>>> s.superclasses(with_self=False).to_set().ids
frozenset({"MS:1000031"})
>>> ms["MS:1000031"]
Term('MS:1000031', name='instrument model')
```

2.4.3 View Collections

The following classes are dedicated collections that are implemented to view a specific field of entities, such as relationships. These types cannot be instantiated directly, but are reachable through the right property on `Entity` instances.

| pronto.entity.attributes. Relationships | A dedicated mutable mapping to manage the relationships of an entity. |

**Relationships**

```python
class pronto.entity.attributes.Relationships
A dedicated mutable mapping to manage the relationships of an entity.

__eq__(other)
Return self==value.

__ge__()
Return self>=value.

__gt__()
Return self>value.

__le__()
Return self<=value.

__lt__()
Return self<value.
```
__repr__()  
Return repr(self).

clear() → None. Remove all items from D.

get(k[, d]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(k[, d]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(k[, d]) → D.get(k,d), also set D[k]=d if k not in D

update([E], **F) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k]
If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v
In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

2.4.4 Model Layer

The following classes are technically part of the data layer, but because they can be lightweight enough to be instantiated directly, they can also be passed to certain functions or properties of the view layer. **Basically, these classes are not worth to implement following the view-model pattern so they can be accessed and mutated directly.**

<table>
<thead>
<tr>
<th>pronto.Metadata</th>
<th>A mapping containing metadata about the current ontology.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pronto.Definition</td>
<td>A human-readable text definition of an entity.</td>
</tr>
<tr>
<td>pronto.Subset</td>
<td>A definition of a subset in an ontology.</td>
</tr>
<tr>
<td>pronto.SynonymType</td>
<td>A user-defined synonym type.</td>
</tr>
<tr>
<td>pronto.LiteralPropertyValue</td>
<td>A property-value which adds a literal annotation to an entity.</td>
</tr>
<tr>
<td>pronto.ResourcePropertyValue</td>
<td>A property-value which adds a resource annotation to an entity.</td>
</tr>
<tr>
<td>pronto.Xref</td>
<td>A cross-reference to another document or resource.</td>
</tr>
</tbody>
</table>

Metadata

**class pronto.Metadata**

A mapping containing metadata about the current ontology.

**format_version**

The OBO format version of the referenced ontology. **1.4** is the default since pronto can only parse and write OBO documents of that format version.

Type `str`

**data_version**

The OBO data version of the ontology, which is then expanded to the versionIRI if translated to OWL.

Type `str` or `None`
ontology
The identifier of the ontology, either as a short OBO identifier or as a full IRI.
Type `str` or `None`

date
The date the ontology was last modified, if any.
Type `datetime` or `None`

default_namespace
The default namespace to use for entity frames lacking a `namespace` clause.
Type `str` or `None`

namespace_id_rule
The production rule for identifiers in the current ontology document. *soft-deprecated, used mostly by OBO-Edit or other outdated tools.*
Type `str` or `None`

owl_axioms
A list of OWL axioms that cannot be expressed in OBO language, serialized in OWL2 Functional syntax.
Type `list of str`

saved_by
The name of the person that last saved the ontology file.
Type `str` or `None`

auto_generated_by
The name of the software that was used to generate the file.
Type `str` or `None`

subsetdefs
A set of ontology subsets declared in the ontology files.
Type `set of str`

imports
A set of references to other ontologies that are imported by the current ontology. OBO requires all entities referenced in the file to be reachable through imports (excluding databases cross-references).
Type `set of str`

synonymtypedefs
A set of user-defined synonym types including a description and an optional scope.
Type `set of SynonymType`

idspaces
A mapping between a local ID space and a global ID space, with an optional description of the mapping.
Type `dict of str` to couple of `str`

remarks
A set of general comments for this file, which will be preserved by a parser/serializer as opposed to inline comments using `!`.
Type `set of str`

annotations
A set of annotations relevant to the whole file. OBO property-values are semantically equivalent to `owl:AnnotationProperty` in OWL2.
Definition

class pronto.Definition
A human-readable text definition of an entity.

Definitions are human-readable descriptions of an entity in the ontology graph, with some optional cross-references to support the definition.

Example

Simply create a Definition instance by giving it a string:

```python
>>> def1 = pronto.Definition('a structural anomaly')
```

Additional cross-references can be passed as arguments, or added later to the xrefs attribute of the Definition:

```python
>>> def2 = pronto.Definition('...', xrefs=(pronto.Xref('MGI:Anna')))
>>> def2.xrefs.add(pronto.Xref('ORCID:0000-0002-3947-4444'))
```

The text content of the definition can be accessed by casting the definition object to a plain string:

```python
>>> str(def1)
'a structural anomaly'
```
**Caution:** A **Definition** compare only based on its textual value, independently of the **Xref** it may contains:

```python
>>> def2 == pronto.Definition('...')
True
```

**Note:** Some ontologies use the xrefs of a description to attribute the authorship of that definition:

```python
>>> cio = pronto.Ontology.from_obo_library("cio.obo")
>>> sorted(cio['CIO:0000011'].definition.xrefs)
[Xref('Bgee:fbb')]
```

The common usecase however is to refer to the source of a definition using persistent identifiers like ISBN book numbers or PubMed IDs.

```python
>>> pl = pronto.Ontology.from_obo_library("plana.obo")
>>> sorted(pl['PLANA:0007518'].definition.xrefs)
[Xref('ISBN:0-71677033-4'), Xref('PMID:4853064')]```

__eq__ ()
Return self==value.

__ge__ ()
Return self>=value.

__gt__ ()
Return self>value.

__hash__ ()
Return hash(self).

__le__ ()
Return self<=value.

__lt__ ()
Return self<value.

__repr__ () → str
Return repr(self).

capitalize ()
Return a capitalized version of the string.

  More specifically, make the first character have upper case and the rest lower case.

casefold ()
Return a version of the string suitable for caseless comparisons.

center ()
Return a centered string of length width.

  Padding is done using the specified fill character (default is a space).

count (sub [, start [, end ]]) → int
Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.
encode ()
Encode the string using the codec registered for encoding.

encoding The encoding in which to encode the string.

ersors The error handling scheme to use for encoding errors. The default is ‘strict’ meaning that encoding
ersors raise a UnicodeEncodeError. Other possible values are ‘ignore’, ‘replace’ and ‘xmlcharrefre-
place’ as well as any other name registered with codecs.register_error that can handle UnicodeEn-
codeErrors.

endswith (suffix[, start[, end]]) → bool
Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at
that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to
try.

expandtabs ()
Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find (sub[, start[, end]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format (*args, **kwargs) → str
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified
by braces (‘{’ and ‘}’).

format_map (mapping) → str
Return a formatted version of S, using substitutions from mapping. The substitutions are identified by
braces (‘{’ and ‘}’).

index (sub[, start[, end]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum ()
Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character
in the string.

isalpha ()
Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the
string.

isascii ()
Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal ()
Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the
string.
isdigit()  
Return True if the string is a digit string, False otherwise.  
A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()  
Return True if the string is a valid Python identifier, False otherwise.  
Use keyword.iskeyword() to test for reserved identifiers such as “def” and “class”.

islower()  
Return True if the string is a lowercase string, False otherwise.  
A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()  
Return True if the string is a numeric string, False otherwise.  
A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()  
Return True if the string is printable, False otherwise.  
A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()  
Return True if the string is a whitespace string, False otherwise.  
A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()  
Return True if the string is a title-cased string, False otherwise.  
In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()  
Return True if the string is an uppercase string, False otherwise.  
A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join()  
Concatenate any number of strings.  
The string whose method is called is inserted in between each given string. The result is returned as a new string.  
Example: ‘.’.join([‘ab’, ‘pq’, ‘rs’]) -> ‘ab.pq.rs’

ljust()  
Return a left-justified string of length width.  
Padding is done using the specified fill character (default is a space).

lower()  
Return a copy of the string converted to lowercase.

lstrip()  
Return a copy of the string with leading whitespace removed.  
If chars is given and not None, remove characters in chars instead.
static maketrans ()
    Return a translation table usable for str.translate().
    
    If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters
to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two
arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be
mapped to the character at the same position in y. If there is a third argument, it must be a string, whose
characters will be mapped to None in the result.

partition ()
    Partition the string into three parts using the given separator.
    
    This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the
    part before the separator, the separator itself, and the part after it.
    
    If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

replace ()
    Return a copy with all occurrences of substring old replaced by new.

    count  Maximum number of occurrences to replace. -1 (the default value) means replace all
    occurrences.

    If the optional argument count is given, only the first count occurrences are replaced.

rfind (sub [, start [, end ]]) → int
    Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
    Optional arguments start and end are interpreted as in slice notation.

    Return -1 on failure.

rindex (sub [, start [, end ]]) → int
    Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
    Optional arguments start and end are interpreted as in slice notation.

    Raises ValueError when the substring is not found.

rjust ()
    Return a right-justified string of length width.

    Padding is done using the specified fill character (default is a space).

rpartition ()
    Partition the string into three parts using the given separator.
    
    This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple
    containing the part before the separator, the separator itself, and the part after it.
    
    If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit ()
    Return a list of the words in the string, using sep as the delimiter string.

    sep  The delimiter according which to split the string. None (the default value) means split ac-
    cording to any whitespace, and discard empty strings from the result.

    maxsplit  Maximum number of splits to do. -1 (the default value) means no limit.

    Splits are done starting at the end of the string and working to the front.

rstrip ()
    Return a copy of the string with trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
split()
Return a list of the words in the string, using sep as the delimiter string.

sep  The delimiter according which to split the string. None (the default value) means split according to
any whitespace, and discard empty strings from the result.

maxsplit  Maximum number of splits to do. -1 (the default value) means no limit.

splitlines()
Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith (prefix[, start[, end]]) → bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at
that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to
try.

strip()
Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase()
Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()
Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower
case.

translate()
Replace each character in the string using the given translation table.

table  Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals,
strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this oper-
ation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

upper()
Return a copy of the string converted to uppercase.

zfill()
Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

Subset

class pronto.Subset
A definition of a subset in an ontology.

name
The name of the subset, as an OBO short identifier.

Type  str

description
A description of the subset, as defined in the metadata part of the ontology file.

Type  str
__eq__ (other: object) → bool
    Return self==value.

__lt__ (other: object) → bool
    Return self<value.

__hash__() → int
    Return hash(self).

__ge__ (other, NotImplemented=NotImplemented)
    Return a >= b. Computed by @total_ordering from (not a < b).

__gt__ (other, NotImplemented=NotImplemented)
    Return a > b. Computed by @total_ordering from (not a < b) and (a != b).

__le__ (other, NotImplemented=NotImplemented)
    Return a <= b. Computed by @total_ordering from (a < b) or (a == b).

__repr__()  
    Return a repr string that roundtrips. Computed by @roundrepr.

**SynonymType**

class pronto.SynonymType
    A user-defined synonym type.

    __eq__(other)
        Return self==value.

    __lt__(other)
        Return self<value.

    __hash__()
        Return hash(self).

    __ge__(other, NotImplemented=NotImplemented)
        Return a >= b. Computed by @total_ordering from (not a < b).

    __gt__(other, NotImplemented=NotImplemented)
        Return a > b. Computed by @total_ordering from (not a < b) and (a != b).

    __le__(other, NotImplemented=NotImplemented)
        Return a <= b. Computed by @total_ordering from (a < b) or (a == b).

    __repr__()
        Return a repr string that roundtrips. Computed by @roundrepr.

**LiteralPropertyValue**

class pronto.LiteralPropertyValue
    A property-value which adds a literal annotation to an entity.

    __eq__(other: object) → bool
        Return self==value.

    __lt__(other: object) → bool
        Return self<value.

    __hash__() → int
        Return hash(self).
__ge__ (other, NotImplemented=NotImplemented)
Return a >= b. Computed by @total_ordering from (not a < b).

__gt__ (other, NotImplemented=NotImplemented)
Return a > b. Computed by @total_ordering from (not a < b) and (a != b).

__le__ (other, NotImplemented=NotImplemented)
Return a <= b. Computed by @total_ordering from (a < b) or (a == b).

__repr__()
Return a repr string that roundtrips. Computed by @roundrepr.

ResourcePropertyValue

class pronto.ResourcePropertyValue
A property-value which adds a resource annotation to an entity.

__eq__ (other: object) → bool
Return self==value.

__ge__ (other, NotImplemented=NotImplemented)
Return a >= b. Computed by @total_ordering from (not a < b).

__gt__ (other, NotImplemented=NotImplemented)
Return a > b. Computed by @total_ordering from (not a < b) and (a != b).

__le__ (other, NotImplemented=NotImplemented)
Return a <= b. Computed by @total_ordering from (a < b) or (a == b).

__repr__()
Return a repr string that roundtrips. Computed by @roundrepr.

__lt__ (other: object) → bool
Return self<value.

__hash__ () → int
Return hash(self).

Xref

class pronto.Xref
A cross-reference to another document or resource.

Cross-references (xrefs for short) can be used to back-up definitions of entities, synonyms, or to link ontological entities to other resources they may have been derived from. Although originally intended to provide links to databases, cross-references in OBO ontologies gained additional purposes, such as helping for header macros expansion, or being used to alias external relationships with local unprefixed IDs.

The OBO format version 1.4 expects references to be proper OBO identifiers that can be translated to actual IRIs, which is a breaking change from the previous format. Therefore, cross-references are encouraged to be given as plain IRIs or as prefixed IDs using an ID from the IDspace mapping defined in the header.

Example
A cross-reference in the Mammalian Phenotype ontology linking a term to some related Web resource:
>>> mp = pronto.Ontology.from_obo_library("mp.obo")
>>> mp["MP:0030151"].name
'abnormal buccinator muscle morphology'
>>> mp["MP:0030151"].xrefs
frozenset({Xref('https://en.wikipedia.org/wiki/Buccinator_muscle')})

Caution: Xref instances compare only using their identifiers; this means it is not possible to have several
cross-references with the same identifier and different descriptions in the same set.

Todo: Make sure to resolve header macros for xrefs expansion (such as treat-xrefs-as-is-a) when
creating an ontology, or provide a method on Ontology doing so when called.

__eq__(other: object) → bool
   Return self==value.
__gt__(other: object) → bool
   Return self>value.
__ge__(other: object) → bool
   Return self>=value.
__lt__(other: object) → bool
   Return self<value.
__le__(other: object) → bool
   Return self<=value.
__repr__() →
   Return a repr string that roundtrips. Computed by @roundrepr.
__hash__() →
   Return hash(self).

2.4.5 Data Layer

The following classes are from the data layer, and store the data extracted from ontology files. There is probably no
point in using them directly, with the exception of custom parser implementations.

<table>
<thead>
<tr>
<th>pronto.RelationshipData</th>
<th>Internal data storage of Relationship information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pronto.SynonymData</td>
<td>Internal data storage of Synonym information.</td>
</tr>
<tr>
<td>pronto.TermData</td>
<td>Internal data storage of Term information.</td>
</tr>
</tbody>
</table>
RelationshipData

class pronto.RelationshipData
   Internal data storage of Relationship information.
       __eq__(self, value)
          Return self==value.
       __ge__(self, value)
          Return self>=value.
       __gt__(self, value)
          Return self>value.
       __hash__(self)
          Return hash(self).
       __le__(self, value)
          Return self<=value.
       __lt__(self, value)
          Return self<value.
       __repr__(self)
          Return repr(self).

SynonymData

class pronto.SynonymData
   Internal data storage of Synonym information.
       __eq__(self, other)
          Return self==value.
       __lt__(self, other)
          Return self<value.
       __hash__(self)
          Return hash(self).
       __ge__(self, other, NotImplemented=NotImplemented)
          Return a >= b. Computed by @total_ordering from (not a < b).
       __gt__(self, other, NotImplemented=NotImplemented)
          Return a > b. Computed by @total_ordering from (not a < b) and (a != b).
       __le__(self, other, NotImplemented=NotImplemented)
          Return a <= b. Computed by @total_ordering from (a < b) or (a == b).
       __repr__(self)
          Return a repr string that roundtrips. Computed by @roundrepr.
TermData

class pronto.TermData
    Internal data storage of Term information.

    __eq__()
        Return self==value.

    __ge__()
        Return self>=value.

    __gt__()
        Return self>value.

    __hash__()
        Return hash(self).

    __le__()
        Return self<=value.

    __lt__()
        Return self<value.

    __repr__()
        Return repr(self).

2.4.6 Warnings

Warnings

exception pronto.warnings.ProntoWarning
    The class for all warnings raised by pronto.

exception pronto.warnings.NotImplementedWarning
    Some part of the code is yet to be implemented.

exception pronto.warnings.SyntaxWarning(*args, **kwargs)
    The parsed document contains incomplete or unsound constructs.

exception pronto.warnings.UnstableWarning
    The behaviour of the executed code might change in the future.

<table>
<thead>
<tr>
<th>pronto.warnings.ProntoWarning</th>
<th>The class for all warnings raised by pronto.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pronto.warnings.NotImplementedWarning</td>
<td>Some part of the code is yet to be implemented.</td>
</tr>
<tr>
<td>pronto.warnings.SyntaxWarning</td>
<td>The parsed document contains incomplete or unsound constructs.</td>
</tr>
<tr>
<td>pronto.warnings.UnstableWarning</td>
<td>The behaviour of the executed code might change in the future.</td>
</tr>
</tbody>
</table>
2.5 To-do

Todo: Make `Term.objects` take in account `holds_over_chain` and `transitive_over` values of the relationship it is building an iterator with.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pronto/checkouts/latest/pronto/term.py:docstring of pronto.Term.objects, line 11.)

Todo: Make sure to resolve header macros for xrefs expansion (such as `treat-xrefs-as-is_a`) when creating an ontology, or provide a method on `Ontology` doing so when called.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pronto/checkouts/latest/pronto/xref.py:docstring of pronto.Xref, line 33.)

2.6 Changelog

All notable changes to this project will be documented in this file.
The format is based on `Keep a Changelog` and this project adheres to `Semantic Versioning`.

2.6.1 Unreleased

2.6.2 2.4.1 - 2021-02-19

Changed

- `pronto.pv.PropertyValue` is now an abstract class.
- `pronto.parsers.RdfXmlParser` now ignores synonym Xrefs not in the right format.

Fixed

- `pronto.Entity.definition` documentation now lists return type as `pronto.definition`. Definition as expected.
- CURIE compaction in RDF/XML not being handled consistently, causing some crashes on ontologies using aliased relationships.
- `pronto.utils.typechecked.disabled` is now reentrant and should not be disabled in multithreaded contexts anymore.
Removed

- Implicit injection of lxml instead of xml.etree, which caused issues with RdfXmlParser.

2.6.3 2.4.0 - 2021-02-18

Added

- Deprecation warnings for the retrieval of relationships via indexing, to be deprecated in v3.

Changed

- Replaced Travis-CI with GitHub Actions to handle continuous integration.
- Bumped fastobo dependency to v0.10.0.

Removed

- Retrieval of terms via their alternate IDs (introduced in v2.3.0, caused multiple issues (#120, #126).

2.6.4 2.3.2 - 2020-12-17

Added

- Support for path-like objects when creating an Ontology (#108).

Fixed

- Avoid decoding file-like objects if they are already UTF-8 encoded when creating an Ontology (#110).

2.6.5 2.3.1 - 2020-09-21

Fixed

- pronto.entity package not being included in source distribution.

2.6.6 2.3.0 - 2020-09-21 - YANKED

Added

- Retrieval of entities via their alternate IDs on the source Ontology.
- Direct edition of entity relationships via the Relationships view.
- __all__ attribute to all modules of the data model.
- RelationshipSet container like TermSet with shortcut attributes and proxying of actual Relationship instances.
• `Relationship.subproperties` and `Relationship.superproperties` methods to add, remove, clear and iterate over the subproperties and superproperties of a `Relationship` instance.
• `Ontology.synonym_types` method to count (via `SizedIterator`) and iterate over the synonym types of an ontology and all of its imports.
• `Ontology.get_synonym_type` method to retrieve a single synonym type by ID from an ontology or one of its imports.

**Changed**

• Management of sub-properties / super-properties is now consistent with the management of subclasses / superclasses.
• `consider`, `disjoint_from`, `disjoint_over`, `equivalent_to`, `replaced_by`, `transitive_over` and `union_of` properties of `Relationship` now return a `RelationshipSet`.

**Fixed**

• Outdated documentation in `Term.subclasses` describing the performances of the previous algorithm.
• Possible `AttributeError` with the setter of the `Entity.synonyms` property.
• Issue with synonym types declared in imported ontologies not being usable with synonyms of the actual ontology.
• Various type annotations not updated since version 2.2.2.

**2.6.7 2.2.4 - 2020-09-11**

**Changed**

• Make `Entity.annotations` return a mutable set and add a setter.

**Fixed**

• `Term.relationship` erroneously updating the `Ontology._lineage` cache.
• Unneeded code in `pronto.serializers._fastobo` handling `is_a` clauses.

**2.6.8 2.2.3 - 2020-07-31**

**Changed**

• Replaced `frozendict` with `immutabledict` (#90).
• Bumped `fastobo` dependency to `v0.9.0` to support inline comments.
• Parsers will now process their imports in parallel using a thread pool.
Fixed

- Argument type checking in view layer is now disabled during the parsing phase to reduce overhead.

2.6.9 2.2.2 - 2020-07-18

Added

- Extraction of basic relationships from RDF/XML documents.

Fixed

- Erroneous type annotations on Term.subclasses and Term.superclasses.
- Bug with Term.equivalent_to setter crashing with a NameError.
- Bug with Entity.synonyms setter not extracting synonym data.

2.6.10 2.2.1 - 2020-06-17

Fixed

- Extraction of subclasses/superclasses hierarchy from nested imports.
- Serialization of OBO frames not being done in order.
- Parsing issue with anti_symmetric clauses in OBO typedefs.
- Xrefs not being extracted when declared as axioms in RDF/XML documents.
- ResourceWarning when creating Ontology from file-handles not mapping to a filesystem location.

2.6.11 2.2.0 - 2020-06-17

Added

- threads parameter to Ontology constructor to control the number of threads used by parsers supporting multithreading (OBO and OBO JSON at the moment).
- Deprecation warnings for suspected uses of the is_a pseudo-relationship since subclasses/superclasses is now to be handled by the owner Ontology.
- Support for subclass/superclass edition directly from the objects returned by Term.subclasses() and Term.superclasses(). (#84)
pronto, Release 2.4.1

Changed

• Updated fastobo to v0.8, which reduce memory footprint of identifiers, and improves the parser speed.
• Improved OBO parser performance using threading plus zero-copy validation of identifiers on Xref instantiation.
• Improved performance in debug mode by having the typechecker only extract the wrapped function signature once.

Fixed

• OBO parser crashing on files containing idspace clauses in their headers.
• Reference management issue with binary operations of TermSet.

Removed

• nanoset dependency, which was not useful anymore in Python 3.8 and caused issues with multithreading when processing OBO frames in parallel.

2.6.12 2.1.0 - 2020-03-23

Added

• Synonym.xrefs now has a setter. (#70)
• pickle support for Ontology. (#66)
• RdfXmlParser support for owl:inverseOf and rdfs:subPropertyOf.

Changed

• Synonym.xrefs now returns a mutable set that can be used to add Xref to the synonym directly.

Fixed

• SynonymType.type setter does not consider all synonym types as undeclared anymore. (#71)
• RdfXmlParser crashing on synonym types definition without a label like in Uberon. (#67)
• FastoboSerializer crashing when encountering a relationship with at least one replaced_by clause.

2.6.13 2.0.1 - 2020-02-19

Fixed

• Internal handling of ontology data forcing an Ontology to outlive all of the Terms created from it.
• Term.id property missing a return type annotation.
• Term.equivalent_to not returning a TermSet but a set of strings.
**Changed**

- Refactored implementation of `SubclassesIterator` and `SuperclassesIterator` to make both use the internal subclassing cache.
- Make `Term.is_leaf` use internal subclassing cache to make it run in constant time.

**2.6.14 2.0.0 - 2020-02-14**

**Added**

- `TermSet.subclasses` and `TermSet.superclasses` methods to query all the subclasses/superclasses of all `Term`.
- `TermSet` class to the top-level `pronto` module.
- Dynamic management of subclassing cache for the `Ontology` class.
- Setters for `Term.consider`, `Term.union_of` and `Term.intersection_of`.

**Removed**

- `cache` keyword argument for the `Ontology`.

**Fixed**

- `SuperclassesIterator.to_set` being named `to_self` because of a typo.
- Several bugs affecting the fastobo-backed serializer.

**2.6.15 1.2.0 - 2020-02-10**

**Added**

- Parameter `with_self` to disable reflexivity of `Term.subclasses` and `Term.superclasses` iterators.
- `TermSet` class which stores a set of terms efficiently while providing some useful shortcuts to access the underlying data.

**Changed**

- Moved code of `Term.subclasses` and `Term.superclasses` to a dedicated iterator class in the `pronto.logic` submodule.
- Dropped `contexter` requirement.
Fixed

• Fix a typo in `Synonym.type` setter leading to a potential bug when the given `type` is `None`.
• Fix miscellaneous bugs found with `mypy`.
• `fastobo` serializer crashing on namespace clauses because of a type issue.
• `fastobo` parsers using data version clauses as format version clauses.

2.6.16  1.1.5 - 2020-01-25

Changed

• Bumped `fastobo` to v0.7.0, switching parser implementation to use multi-threading in order to speedup the parser process.

2.6.17  1.1.4 - 2020-01-21

Added

• Explicit support for Python 3.8.
• Support for Windows-style line endings (#53)

2.6.18  1.1.3 - 2019-11-10

Fixed

• Handling of some clauses in `FastoboParser`.
• `OboSerializer` occasionally missing lines between term and typedef frames.

Added

• Missing docstrings to some `Entity` properties.

2.6.19  1.1.2 - 2019-10-30

Fixed

• `RdfXMLParser` crashing on entities with `rdf:label` elements without literal content.
2.6.20 1.1.1 - 2019-10-29

Fixed

- pronto.serializers module not being embedded in Wheel distribution.

2.6.21 1.1.0 - 2019-10-24

Added

- Entity.add_synonym method to create a new synonym and add it to an entity.
- @roundrepr now adds a minimal docstring to the generated __repr__ method.
- Ontology caches subclassing relationships to greatly improve performance of Term.subclasses.

Changed

- Entity subclasses now store their id directly to improve performance.
- Term.subclasses and Term.superclasses use collections.deque instead of queue.Queue as a LIFO structure since thread-safety is not needed.
- chardet result is now used even when prediction confidence is under 100% to detect encoding of the handle passed to Ontology.

Fixed

- SynonymType comparison implementation.
- Synonym.type getter crashing on type not being None.
- RdfXMLParser crashing on synonymtypedefs without scope specifiers.

2.6.22 1.0.0 - 2019-10-11

Fixed

- Issues with typedef serialization in FastoboSerializer.
- Ontology.create_term and Ontology.create_relationship not raising ValueError when given an identifier already in the knowledge graph.
- Signature of BaseSerializer.dump to remove encoding argument.
- Missing __slots__ in Entity in non-typechecking runtime.


pronto, Release 2.4.1

Changed

- Bumped fastobo requirement to v0.6.0.

2.6.23 1.0.0-alpha.3 - 2019-10-10

Added

- Extraction of oboInOwl:consider annotation in RdfXMLParser.
- Extraction of oboInOwl:savedBy annotation in RdfXMLParser.
- Extraction of subsetdef and synonymtypedef as annotation properties in RdfXMLParser.
- Proper comparison of PropertyValue classes, based on the lexicographic order of their serialization.
- Ontology.dump and Ontology.dumps methods to serialize an ontology in obo or obojson format.

Fixed

- Metadata not storing optional description of ID spaces if any.
- Wrong type hints in RelationshipData.equivalent_to_chain.

Changed

- Added type checking to some more property setters.
- Avoid using networkx in Term.subclasses.
- fastobo-derived parsers will not create a new entity if one exists in the graph of dependencies already.
- Exposed pronto.warnings and the complete warnings hierarchy.

2.6.24 1.0.0-alpha.2 - 2019-10-03

Added

- Support for extraction of relationships from OWL/XML files to OwlXMLParser.

Fixed

- Type hints of RelationshipData.synonyms attribute.
2.6.25 1.0.0-alpha.1 - 2019-10-02

**Changed**

- Dropped support for Python earlier than 3.6.
- Brand new data model that follow the OBO 1.4 object model.
- Partial OWL XML parser implementation using the OBO 1.4 semantics.
- New OBO parser implementation based on fastobo.
- Imports are properly separated from the top-level ontology.
- `Ontology.__getitem__` can also access entities from imports.
- `Term, Relationship, Xref, SynonymType` compare only based on their ID.
- `Subset, Definition` compare only based on their textual value.

**Added**

- Support for OBO JSON parser based on fastobo.
- Provisional `mypy` type hints.
- Type checking for most properties in `__debug__` mode.
- Proper `repr` implementation that should roundtrip most of the time.
- Detection of file format and encoding based on buffer content.

**Removed**

- OBO and JSON serialization support (for now).
- `Term.rchildren` and `Term.rparents` and stop making direction assumptions on relationships.

2.7 About

2.7.1 Authors

`pronto` is developped and maintained by:
2.7.2 Contributors

The following developers contributed both code and time to this project:

- Alex Henrie (@alexhenrie)
- Spencer Mitchell (@smitchell556)
- Tatsuya Sakaguchi (@ttyskg)
- Philipp A. (@flying-sheep)

2.7.3 Reference

If you wish to cite this library, please make sure you are using the latest version of the code, and use the DOI shown on the Zenodo record.
CHAPTER
THREE

INDICES AND TABLES

- genindex
- search
p
pronto, 10
Symbols

__bool__() (pronto.Metadata method), 35
__eq__() (pronto.Definition method), 36
__eq__() (pronto.Entity method), 15
__eq__() (pronto.LiteralPropertyValue method), 41
__eq__() (pronto.Metadata method), 35
__eq__() (pronto.Ontology method), 12
__eq__() (pronto.Relationship method), 19
__eq__() (pronto.RelationshipData method), 44
__eq__() (pronto.ResourcePropertyValue method), 42
__eq__() (pronto.Subset method), 40
__eq__() (pronto.Synonym method), 24
__eq__() (pronto.SynonymData method), 44
__eq__() (pronto.SynonymType method), 41
__eq__() (pronto.Term method), 25
__eq__() (pronto.TermData method), 45
__eq__() (pronto.TermSet method), 31
__eq__() (pronto.Xref method), 43
__eq__() (pronto.entity.attributes.Relationships method), 32
__ge__() (pronto.Definition method), 36
__ge__() (pronto.Entity method), 15
__ge__() (pronto.LiteralPropertyValue method), 41
__ge__() (pronto.Metadata method), 35
__ge__() (pronto.Ontology method), 13
__ge__() (pronto.Relationship method), 19
__ge__() (pronto.RelationshipData method), 44
__ge__() (pronto.ResourcePropertyValue method), 42
__ge__() (pronto.Subset method), 41
__ge__() (pronto.Synonym method), 24
__ge__() (pronto.SynonymData method), 44
__ge__() (pronto.SynonymType method), 41
__ge__() (pronto.Term method), 25
__ge__() (pronto.TermData method), 45
__ge__() (pronto.TermSet method), 31
__ge__() (pronto.Xref method), 43
__ge__() (pronto.entity.attributes.Relationships method), 32
__gt__() (pronto.Metadata method), 35
__gt__() (pronto.Ontology method), 13
__gt__() (pronto.Relationship method), 19
__gt__() (pronto.RelationshipData method), 44
__gt__() (pronto.ResourcePropertyValue method), 42
__gt__() (pronto.Subset method), 41
__gt__() (pronto.Synonym method), 24
__gt__() (pronto.SynonymData method), 44
__gt__() (pronto.SynonymType method), 41
__gt__() (pronto.Term method), 25
__gt__() (pronto.TermData method), 45
__gt__() (pronto.TermSet method), 31
__gt__() (pronto.Xref method), 43
__gt__() (pronto.entity.attributes.Relationships method), 32

__hash__() (pronto.Definition method), 36
__hash__() (pronto.Entity method), 15
__hash__() (pronto.LiteralPropertyValue method), 41
__hash__() (pronto.Metadata method), 35
__hash__() (pronto.Relationship method), 19
__hash__() (pronto.RelationshipData method), 44
__hash__() (pronto.ResourcePropertyValue method), 42
__hash__() (pronto.Subset method), 41
__hash__() (pronto.Synonym method), 24
__hash__() (pronto.SynonymData method), 44
__hash__() (pronto.SynonymType method), 41
__hash__() (pronto.Term method), 25
__hash__() (pronto.TermData method), 45
__hash__() (pronto.TermSet method), 31
__hash__() (pronto.Xref method), 43

__getitem__() (pronto.meta import pronto.Entity method, 15
__getitem__() (pronto.Ontology method), 12
__getitem__() (pronto.Definition method), 36
__getitem__() (pronto.LiteralPropertyValue method), 42

INDEX
description (pronto.Subset attribute), 40
discard() (pronto.TermSet method), 31
disjoint_from() (pronto.Entity property), 17
disjoint_from() (pronto.Relationship property), 21
disjoint_from() (pronto.Term property), 27
disjoint_over() (pronto.Relationship property), 23
domain() (pronto.Relationship property), 23
dump() (pronto.Ontology method), 13
dumps() (pronto.Ontology method), 13

E

encode() (pronto.Definition method), 36
endswith() (pronto.Definition method), 37
Entity (class in pronto), 15
equivalent_to() (pronto.Entity property), 17
equivalent_to() (pronto.Relationship property), 21
equivalent_to() (pronto.Term property), 27
expandtabs() (pronto.Definition method), 37

F

find() (pronto.Definition method), 37
format() (pronto.Definition method), 37
format_map() (pronto.Definition method), 37
format_version (pronto.Metadata attribute), 33
from_obo_library() (pronto.Ontology class method), 11
functional() (pronto.Relationship property), 23

G

get() (pronto.entity.attributes.Relationships method), 33
get() (pronto.Ontology method), 13
get_relationship() (pronto.Ontology method), 14
get_synonym_type() (pronto.Ontology method), 14
get_term() (pronto.Ontology method), 14

H

holds_over_chain() (pronto.Relationship property), 24

I

id() (pronto.Entity property), 17
id() (pronto.Relationship property), 21
id() (pronto.Term property), 27
idspace() (pronto.Metadata attribute), 34
imports (pronto.Metadata attribute), 34
imports (pronto.Ontology attribute), 11
index() (pronto.Definition method), 37
intersection_of() (pronto.Relationship property), 24
intersection_of() (pronto.Term property), 30
inverse_functional() (pronto.Term property), 23

N

name (pronto.Subset attribute), 40
name() (pronto.Entity property), 17
name() (pronto.Relationship property), 21
name() (pronto.Term property), 27
namespace() (pronto.Entity property), 17
namespace() (pronto.Relationship property), 21
namespace() (pronto.Term property), 27
namespace_id_rule (pronto.Metadata attribute), 34
NotImplementedWarning, 45

O

objects() (pronto.Term method), 28
obsolete() (pronto.Entity property), 17
obsolete() (pronto.Relationship property), 21
obsolete() (pronto.Term property), 27
Ontology (class in pronto), 11
ontology (pronto.Metadata attribute), 33
owl_axioms (pronto.Metadata attribute), 34
relationships() (pronto.entity.attributes.Relationships method), 33
reflexive() (pronto.Relationship property), 24
range() (popitem()), 33
pronto (module), 10
ProntoWarning, 45

P
partition() (pronto.Definition method), 39
pop() (pronto.entity.attributes.Relationships method), 33
pop() (pronto.TermSet method), 31
popitem() (pronto.entity.attributes.Relationships method), 33
pronto (module), 10
ProntoWarning, 45

R
range() (pronto.Relationship property), 24
reflexive() (pronto.Relationship property), 24
Relationship (class in pronto), 19
RelationshipData (class in pronto), 44
Relationships (class in pronto.entity.attributes), 32
relationships() (pronto.Ontology method), 14
remarks (pronto.Metadata attribute), 34
remove() (pronto.TermSet method), 31
replace() (pronto.Definition method), 39
replaced_by() (pronto.Entity property), 18
replaced_by() (pronto.Relationship property), 22
replaced_by() (pronto.Term property), 28
ResourcePropertyValue (class in pronto), 42
rfind() (pronto.Definition method), 39
rindex() (pronto.Definition method), 39
rjust() (pronto.Definition method), 39
rpartition() (pronto.Definition method), 39
rsplit() (pronto.Definition method), 39
rstrip() (pronto.Definition method), 39

S
saved_by (pronto.Metadata attribute), 34
setdefault() (pronto.entity.attributes.Relationships method), 33
split() (pronto.Definition method), 39
splitwith() (pronto.Definition method), 40
strip() (pronto.Definition method), 40
subclasses() (pronto.Term method), 29
subclasses() (pronto.TermSet method), 31
superclasses() (pronto.Relationship property), 29
superclasses() (pronto.Term method), 32
superproperties() (pronto.Relationship method), 22
swapcase() (pronto.Definition method), 40
symmetric() (pronto.Relationship property), 24
Synonym (class in pronto), 44
synonym_types() (pronto.Ontology method), 13
SynonymData (class in pronto), 44
synonyms() (pronto.Entity property), 18
synonyms() (pronto.Relationship property), 22
synonyms() (pronto.Term property), 28
SynonymType (class in pronto), 41
synonymtypedefs (pronto.Metadata attribute), 34
SyntaxWarning, 45

T
Term (class in pronto), 25
TermData (class in pronto), 45
terms() (pronto.Ontology method), 13
TermSet (class in pronto), 31
timeout (pronto.Ontology attribute), 11
title() (pronto.Definition method), 40
transitive() (pronto.Relationship property), 24
transitive_over() (pronto.Relationship property), 24
translate() (pronto.Definition method), 40

U
unreserved (pronto.Metadata attribute), 35
UnstableWarning, 45
update() (pronto.entity.attributes.Relationships method), 33
upper() (pronto.Definition method), 40

V
values() (pronto.entity.attributes.Relationships method), 33
values() (pronto.Ontology method), 13

X
Xref (class in pronto), 42
xrefs() (pronto.Entity property), 18
xrefs() (pronto.Relationship property), 22
xrefs() (pronto.Term property), 28

Z
zfill() (pronto.Definition method), 40