

---

# OWLAPY

*Release 1.3.3*

**Ontolearn Team**

**Nov 25, 2024**

## Contents:

<b>1</b>	<b>About owlapy</b>	<b>2</b>
1.1	What is owlapy? . . . . .	2
1.2	What does owlapy have to offer? . . . . .	2
1.3	How to install? . . . . .	3
<b>2</b>	<b>Basic Usage</b>	<b>3</b>
2.1	Atomic Classes . . . . .	3
2.2	Object Property . . . . .	4
2.3	Complex class expressions . . . . .	4
2.4	Convert to SPARQL, DL or Manchester syntax . . . . .	5
<b>3</b>	<b>Ontologies</b>	<b>5</b>
3.1	Loading an Ontology . . . . .	5
3.2	Modifying an Ontology . . . . .	6
3.3	Save an Ontology . . . . .	8
3.4	Worlds . . . . .	8
<b>4</b>	<b>Reasoners</b>	<b>8</b>
4.1	Usage of the Reasoner . . . . .	9
4.2	Class Reasoning . . . . .	9
4.3	Object Properties and Data Properties Reasoning . . . . .	10
4.4	Find Instances . . . . .	11
<b>5</b>	<b>Owlapi Synchronization</b>	<b>11</b>
5.1	“Sync” Classes . . . . .	11
5.2	Notes . . . . .	12
5.3	Examples . . . . .	13
<b>6</b>	<b>Further Resources</b>	<b>13</b>
6.1	More Inside the Project . . . . .	13
6.2	Contribution . . . . .	13
6.3	Questions . . . . .	13
6.4	Coverage Report . . . . .	13
<b>7</b>	<b>owlapy</b>	<b>15</b>
7.1	Submodules . . . . .	15
7.2	Classes . . . . .	154

7.3	Functions	155
7.4	Package Contents	155
<b>Python Module Index</b>		<b>157</b>
<b>Index</b>		<b>158</b>

---

OWLAPY<sup>1</sup>: Representation of OWL objects in python.

## 1 About owlapy

**Version:** owlapy 1.3.3

**GitHub repository:** <https://github.com/dice-group/owlapy>

**Publisher and maintainer:** DICE<sup>2</sup> - data science research group of Paderborn University<sup>3</sup>.

**Contact:** [onto-learn@lists.uni-paderborn.de](mailto:onto-learn@lists.uni-paderborn.de)

**License:** MIT License

---

### 1.1 What is owlapy?

Owlapy is an open-source software library in python that is used to represent entities in OWL 2 Web Ontology Language.

We identified the gap of having a library that will serve as a base structure for representing OWL entities and for manipulating OWL Ontologies in python, and like that, owlapy was created. Owlapy is loosely based on its java-counterpart, *owlapi*. Owlapy is currently utilized by powerful libraries such as [Ontolearn](#)<sup>4</sup> and [OntoSample](#)<sup>5</sup>.

Owlapy is the perfect choice for machine learning projects that are built in python and focus on knowledge graphs and class expression learnings.

---

### 1.2 What does owlapy have to offer?

- Create, manipulate and save Ontologies.
- Retrieving information from the signature of the ontology.
- Reasoning over ontology.
- Represent every notation in [OWL 2 Structural Specification and Functional-Style Syntax](#)<sup>6</sup> including:
  - Entities, Literals, and Anonymous Individuals
  - Property Expressions
  - Data Ranges
  - Class Expressions

---

<sup>1</sup> <https://github.com/dice-group/owlapy>

<sup>2</sup> <https://dice-research.org/>

<sup>3</sup> <https://www.uni-paderborn.de/en/university>

<sup>4</sup> <https://github.com/dice-group/Ontolearn>

<sup>5</sup> <https://github.com/alkidbaci/OntoSample>

<sup>6</sup> <https://www.w3.org/TR/owl2-syntax/>

- Axioms
- Annotations
- Construct complex class expressions.
- Provide interfaces for OWL Ontology, Ontology manager and Reasoner.
- Convert owl expression to SPARQL queries.
- Render owl expression to Description Logics or Manchester syntax.
- Parse Description Logics or Manchester expression to owl expression.

## 1.3 How to install?

Installation from source:

```
git clone https://github.com/dice-group/owlapy
conda create -n temp_owlapy python=3.10.13 --no-default-packages && conda activate_
↪temp_owlapy && pip3 install -e .
```

or using PyPI:

```
pip3 install owlapy
```

## 2 Basic Usage

The main usage for owlapy is to use it for class expression construction. Class expression learning algorithms require such basic structure to work upon. Let's walk through an example of constructing some class expressions.

In this example we will be using the *family* ontology, a simple ontology with namespace: `http://example.com/family#`. Here is a hierarchical diagram that shows the classes and their relationship:

```

      Thing
      |
    person
    /   |
  male female

```

It contains only one object property which is `hasChild` and in total there are six persons (individuals), of which four are males and two are females.

### 2.1 Atomic Classes

To represent the classes `male`, `female`, and `person` we can simply use the class `OWLClass`<sup>7</sup>:

```
from owlapy.class_expression import OWLClass
from owlapy.iri import IRI

namespace = "http://example.com/family#"

male = OWLClass(IRI(namespace, "male"))
female = OWLClass(IRI(namespace, "female"))
```

(continues on next page)

<sup>7</sup> [https://dice-group.github.io/owlapy/autoapi/owlapy/class\\_expression/owl\\_class/index.html#owlapy.class\\_expression.owl\\_class.OWLClass](https://dice-group.github.io/owlapy/autoapi/owlapy/class_expression/owl_class/index.html#owlapy.class_expression.owl_class.OWLClass)

(continued from previous page)

```
person = OWLClass(URI(namespace, "person"))
```

Notice that we created an `URI` object for every class. `URI`<sup>8</sup> is used to represent an *IRI*. Every named entity requires an *IRI*, whereas Anonymous entities does not. However, in owlapy you can create an *OWLClass* by passing the *IRI* directly as a string, like so:

```
male = OWLClass("http://example.com/family#male")
```

## 2.2 Object Property

To represent the object property `hasChild` we can use the class `OWLObjectProperty`<sup>9</sup>:

```
from owlapy.owl_property import OWLObjectProperty

hasChild = OWLObjectProperty("http://example.com/family#hasChild")
```

**Tip:** In owlapy the naming of the classes is made in accordance with the notations from OWL 2 specification but with the word “OWL” in the beginning. Example: “*OWLObjectProperty*” represents the notation “*ObjectProperty*”.

## 2.3 Complex class expressions

Now that we have these atomic entities, we can construct more complex class expressions. Let’s say we want to represent all individuals which are `male` and have at least 1 child.

We already have the concept of `male`. We need to find the appropriate class for the second part: “*have at least 1 child*”. In OWL 2 specification that would be `ObjectMinCardinality`<sup>10</sup>. In owlapy, as we said, we simply add the word “OWL” upfront to find the correct class:

```
from owlapy.class_expression import OWLObjectMinCardinality

has_at_least_one_child = OWLObjectMinCardinality(
    cardinality = 1,
    property = hasChild,
    filler = person
)
```

As you can see, to create an object of class `OWLObjectMinCardinality`<sup>11</sup> is as easy as that. You specify the cardinality which in this case is 1, the object property where we apply this cardinality restriction and the filler class in case you want to restrict the domain of the class expression. In this case we used `person`.

Now let’s merge both class expressions together using `OWLObjectIntersectionOf`<sup>12</sup>:

```
from owlapy.class_expression import OWLObjectIntersectionOf

ce = OWLObjectIntersectionOf([male, has_at_least_one_child])
```

<sup>8</sup> <https://dice-group.github.io/owlapy/autoapi/owlapy/iri/index.html#owlapy.iri.URI>

<sup>9</sup> [https://dice-group.github.io/owlapy/autoapi/owlapy/owl\\_property/index.html#owlapy.owl\\_property.OWLObjectProperty](https://dice-group.github.io/owlapy/autoapi/owlapy/owl_property/index.html#owlapy.owl_property.OWLObjectProperty)

<sup>10</sup> [https://www.w3.org/TR/owl2-syntax/#Minimum\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Minimum_Cardinality)

<sup>11</sup> [https://dice-group.github.io/owlapy/autoapi/owlapy/class\\_expression/restriction/index.html#owlapy.class\\_expression.restriction.OWLObjectMinCardinality](https://dice-group.github.io/owlapy/autoapi/owlapy/class_expression/restriction/index.html#owlapy.class_expression.restriction.OWLObjectMinCardinality)

<sup>12</sup> [https://dice-group.github.io/owlapy/autoapi/owlapy/class\\_expression/nary\\_boolean\\_expression/index.html#owlapy.class\\_expression.nary\\_boolean\\_expression.OWLObjectIntersectionOf](https://dice-group.github.io/owlapy/autoapi/owlapy/class_expression/nary_boolean_expression/index.html#owlapy.class_expression.nary_boolean_expression.OWLObjectIntersectionOf)

## 2.4 Convert to SPARQL, DL or Manchester syntax

Owlapy is not just a library to represent OWL entities, you can also use it to convert owl expressions into other formats:

```
from owlapy import owl_expression_to_sparql, owl_expression_to_dl, owl_expression_to_
↳manchester

print(owl_expression_to_dl(ce))
# Result: male ♂ (≥ 1 hasChild.person)

print(owl_expression_to_sparql(ce))
# Result: SELECT DISTINCT ?x WHERE { ?x a <http://example.com/family#male> . { SELECT_
↳?x WHERE { ?x <http://example.com/family#hasChild> ?s_1 . ?s_1 a <http://example.
↳com/family#person> . } GROUP BY ?x HAVING ( COUNT ( ?s_1 ) >= 1 ) } }

print(owl_expression_to_manchester(ce))
# Result: male and (hasChild min 1 person)
```

To parse a DL or Manchester expression to owl expression you can use the following convenient methods:

```
from owlapy import dl_to_owl_expression, manchester_to_owl_expression

print(dl_to_owl_expression("∃ hasChild.male", namespace))
# Result: OWLObjectSomeValuesFrom(property=OWLObjectProperty(IRI('http://example.com/
↳family#', 'hasChild')),filler=OWLObject(IRI('http://example.com/family#', 'male')))

print(manchester_to_owl_expression("female and (hasChild max 2 person)", namespace))
# Result: OWLObjectIntersectionOf((OWLObject(IRI('http://example.com/family#', 'female
↳')), OWLObjectMaxCardinality(property=OWLObjectProperty(IRI('http://example.com/
↳family#', 'hasChild')),2,filler=OWLObject(IRI('http://example.com/family#', 'person
↳')))))
```

In these examples we showed a fraction of **owlapy**. You can explore the *api documentation* to learn more about all classes in owlapy and check more examples in the `examples`<sup>13</sup> directory.

## 3 Ontologies

To get started with Structured Machine Learning, the first thing required is an *Ontology*<sup>14</sup> with *Named Individuals*<sup>15</sup>. In this guide we show the basics of working with ontologies in Owlapy. We will use the *father* ontology for the following examples.

### 3.1 Loading an Ontology

To load an ontology as well as to manage it, you will need an *OWLOntologyManager*. An ontology can be loaded using the following Python code:

```
from owlapy.iri import IRI
from owlapy.owl_ontology_manager import OntologyManager
```

(continues on next page)

<sup>13</sup> <https://github.com/dice-group/owlapy/tree/develop/examples>

<sup>14</sup> <https://www.w3.org/TR/owl2-overview/>

<sup>15</sup> [https://www.w3.org/TR/owl-syntax/#Named\\_Individuals](https://www.w3.org/TR/owl-syntax/#Named_Individuals)

(continued from previous page)

```
manager = OntologyManager()
onto = manager.load_ontology(IRI.create("file://KGs/Family/father.owl"))
```

First, we import the `IRI` class and a suitable `OWL OntologyManager`. To load a file from our computer, we have to reference it with an *IRI*. Secondly, we need the Ontology Manager. Owlapy contains one such manager: The *Ontology-Manager*.

Now, we can already inspect the contents of the ontology. For example, to list all individuals:

```
for ind in onto.individuals_in_signature():
    print(ind)
```

You can get the object properties in the signature:

```
onto.object_properties_in_signature()
```

For more methods, see the abstract class *OWLOntology* or the concrete implementation *Ontology*.

## 3.2 Modifying an Ontology

Axioms in ontology serve as the basis for defining the vocabulary of a domain and for making statements about the relationships between individuals and concepts in that domain. They provide a formal and precise way to represent knowledge and allow for automated reasoning and inference. Axioms can be **added**, **modified**, or **removed** from an ontology, allowing the ontology to evolve and adapt as new knowledge is gained.

In owlapy we also have different axioms represented by different classes. You can check all the axioms classes *here*. Some frequently used axioms are:

- *OWLDeclarationAxiom*
- *OWLObjectPropertyAssertionAxiom*
- *OWLDataPropertyAssertionAxiom*
- *OWLClassAssertionAxiom*
- *OWLSubClassOfAxiom*
- *OWLEquivalentClassesAxiom*

### Add a new Class

Let's suppose you want to add a new class in our example ontology `KGs/Family/father.owl`. It can be done as follows:

```
from owlapy.class_expression import OWLClass
from owlapy.owl_axiom import OWLDeclarationAxiom

iri = IRI('http://example.com/father#', 'child')
child_class = OWLClass(iri)
child_class_declaration_axiom = OWLDeclarationAxiom(child_class)

onto.add_axiom(child_class_declaration_axiom)
```

In this example, we added the class 'child' to the *father.owl* ontology. Firstly we create an instance of *OWLClass* to represent the concept of 'child' by using an *IRI*. On the other side, an instance of `IRI` is created by passing two arguments which are the namespace of the ontology and the remainder 'child'. To declare this new class we need an axiom of type *OWLDeclarationAxiom*. We simply pass the `child_class` to create an instance of this axiom. The final

step is to add this axiom to the ontology. We use the `add_axiom` method to add into the ontology onto the axiom `child_class_declaration_axiom`.

## Add a new Object Property / Data Property

The idea is the same as adding a new class. Instead of `OWLClass`, for object properties, you can use the class *OWLObjectProperty* and for data properties you can use the class *OWLDataProperty*.

```
from owlapy.owl_property import OWLObjectProperty, OWLDataProperty

# adding the object property 'hasParent'
hasParent_op = OWLObjectProperty(URI('http://example.com/father#', 'hasParent'))
hasParent_op_declaration_axiom = OWLDeclarationAxiom(hasParent_op)
onto.add_axiom(hasParent_op_declaration_axiom)

# adding the data property 'hasAge'
hasAge_dp = OWLDataProperty(URI('http://example.com/father#', 'hasAge'))
hasAge_dp_declaration_axiom = OWLDeclarationAxiom(hasAge_dp)
onto.add_axiom(hasAge_dp_declaration_axiom)
```

See the *owlapy* for more OWL entities that you can add as a declaration axiom.

## Add an Assertion Axiom

To assign a class to a specific individual use the following code:

```
from owlapy.owl_axiom import OWLClassAssertionAxiom

individuals = list(onto.individuals_in_signature())
heinz = individuals[1] # get the 2nd individual in the list which is 'heinz'

class_assertion_axiom = OWLClassAssertionAxiom(heinz, child_class)

onto.add_axiom(class_assertion_axiom)
```

We have used the previous method `individuals_in_signature()` to get all the individuals and converted them to a list, so we can access them by using indexes. In this example, we want to assert a class axiom for the individual `heinz`. We have used the class `OWLClassAssertionAxiom` where the first argument is the 'individual' `heinz` and the second argument is the 'class\_expression'. As the class expression, we used the previously defined class `child_Class`. Finally, add the axiom by using `add_axiom` method of the *OWLOntology*.

Let's show one more example using a `OWLDataPropertyAssertionAxiom` to assign the age of 17 to `heinz`.

```
from owlapy.owl_literal import OWLLiteral
from owlapy.owl_axiom import OWLDataPropertyAssertionAxiom

literal_17 = OWLLiteral(17)
dp_assertion_axiom = OWLDataPropertyAssertionAxiom(heinz, hasAge_dp, literal_17)

onto.add_axiom(dp_assertion_axiom)
```

*OWLLiteral* is a class that represents the literal values in Owlapy. We have stored the integer literal value of '17' in the variable `literal_17`. Then we construct the `OWLDataPropertyAssertionAxiom` by passing as the first argument, the individual `heinz`, as the second argument the data property `hasAge_dp`, and the third argument the literal value `literal_17`. Finally, add it to the ontology by using `add_axiom` method.

Check the [owlapy](#) to see all the OWL assertion axioms that you can use.

## Remove an Axiom

To remove an axiom you can use the `remove_axiom` method as follows:

```
onto.remove_axiom(dp_assertion_axiom)
```

The required argument is the axiom/axioms you want to remove.

## 3.3 Save an Ontology

If you modified an ontology, you may want to save it as a new file. To do this you can use the `save` method of the *OWLOntology*. It requires one argument, the IRI of the new ontology.

```
onto.save(IRI.create('file://' + 'test' + '.owl'))
```

The above line of code will save the ontology `onto` in the file `test.owl` which will be created in the same directory as the file you are running this code.

## 3.4 Worlds

Owlready2 stores every triple in a ‘World’ object, and it can handle several Worlds in parallel. Owlready2 uses an optimized quadstore to store the world. Each world object is stored in a separate quadstore and by default the quadstore is stored in memory, but it can also be stored in an SQLite3 file. The method `save_world()` of the ontology manager does the latter. When an *OWLOntologyManager* object is created, a new world is also created as an attribute of the manager. By calling the method `load_ontology(iri)` the ontology is loaded to this world.

It is possible to create several isolated “worlds”, sometimes called “universe of speech”. This makes it possible, in particular, to load the same ontology several times, independently, that is to say, without the modifications made on one copy affecting the other copy. Sometimes the need to isolate an ontology arises. What that means is that you can have multiple references of the same ontology in different worlds.

---

It is important that an ontology is associated with a reasoner which is used to infer knowledge from the ontology, i.e. to perform ontology reasoning. In the next guide we will see how to use a reasoner in Owlapy.

## 4 Reasoners

To validate facts about statements in the ontology, the help of a reasoner component is required.

For this guide we will also consider the ‘father’ ontology that we slightly described [here](#):

```
from owlapy.iri import IRI
from owlapy.owl_ontology_manager import OntologyManager

manager = OntologyManager()
onto = manager.load_ontology(IRI.create("KGs/Family/father.owl"))
```

In our Owlapy library, we provide two main reasoner classes:

- **StructuralReasoner** (What used to be FastInstanceCheckerReasoner)

Structural Reasoner is the base reasoner in Owlapy. This reasoner works under CWA/PCWA and the base library used for it is *owlready2*. The functionalities of this reasoner are limited and may be incomplete. It does not provide



full reasoning in *ALCH*. It provides support for finding instance of complex class expression. It has a cache storing system that allows for faster execution of some reasoning functionalities.

#### Initialization:

```
from owlapy.owl_reasoner import StructuralReasoner

structural_reasoner = StructuralReasoner(onto, property_cache = True, negation_
↪ default = True, sub_properties = False)
```

The structural reasoner requires an ontology (*AbstractOWLOntology*). `property_cache` specifies whether to cache property values. This requires more memory, but it speeds up the reasoning processes. If `negation_default` argument is set to `True` the missing facts in the ontology means false. The argument `sub_properties` is another boolean argument to specify whether you want to take sub properties in consideration for `instances()` method.

- *SyncReasoner*

*SyncReasoner* is a class that serves as a ‘syncing’ class between our framework and reasoners in *owlapi*. It can perform full reasoning in *ALCH* due to the use of reasoners from powerful reasoners like *HermiT*, *Pellet*, etc. *SyncReasoner* is more useful when your main goal is reasoning over the ontology, and you are familiarized with the java reasoners (*HermiT*, *Pellet*, *JFact*, *Openllet*, ...).

#### Initialization:

```
from owlapy.owl_reasoner import SyncReasoner

sync_reasoner = SyncReasoner(ontology="KGs/Mutagenesis/mutagenesis.owl", reasoner=
↪ "HermiT")
```

*SyncReasoner* is made available by *owlapi mapper* and requires the ontology path or an object of type *SyncOntology*, together with a reasoner name from the possible set of reasoners: "HermiT", "Pellet", "JFact", "Openllet" "StructuralReasoner".

**Note that *SyncReasoner* with `reasoner` argument set to "StructuralReasoner" is referring to *StructuralReasoner* implemented in *owlapi*. That is different from our *StructuralReasoner*.**

## 4.1 Usage of the Reasoner

All the reasoners available in the Owlapy library inherit from the class: *AbstractOWLReasoner*. Further on, in this guide, we use *StructuralReasoner* to show the capabilities of a reasoner in Owlapy.

We will proceed to use the *father* dataset to give examples.

## 4.2 Class Reasoning

Using an *AbstractOWLOntology* you can list all the classes in the signature, but a reasoner can give you more than that. You can get the subclasses, superclasses or the equivalent classes of a class in the ontology:

```
from owlapy.class_expression import OWLClass
from owlapy.iri import IRI

namespace = "http://example.com/father#"
male = OWLClass(IRI(namespace, "male"))

male_super_classes = structural_reasoner.super_classes(male)
```

(continues on next page)

(continued from previous page)

```
male_sub_classes = structural_reasoner.sub_classes(male)
male_equivalent_classes = structural_reasoner.equivalent_classes(male)
```

We define the *male* class by creating an *OWLClass* object. The methods *super\_classes* and *sub\_classes* have 2 more boolean arguments: *direct* and *only\_named*. If *direct=True* then only the direct classes in the hierarchy will be returned, else it will return every class in the hierarchy depending on the method (*sub\_classes* or *super\_classes*). By default, its value is *False*. The next argument *only\_named* specifies whether you want to show only named classes or complex classes as well. By default, its value is *True* which means that it will return only the named classes.

**NOTE:** The extra arguments *direct* and *only\_named* are also used in other methods that reason upon the class, object property, or data property hierarchy.

**NOTE:** In *SyncReasoner*, there is no use for the argument *only\_named* as this is not supported by methods in the java library *owlapi*.

You can get all the types of a certain individual using *types* method:

```
anna = list(onto.individuals_in_signature()).pop(0)

anna_types = structural_reasoner.types(anna)
```

We retrieve *anna* as the first individual on the list of individuals of the 'Father' ontology. The *types* method only returns named classes.

### 4.3 Object Properties and Data Properties Reasoning

Owlapy reasoners offers some convenient methods for working with object properties and data properties. Below we show some of them, but you can always check all the methods in the *AbstractOWLReasoner* class documentation.

You can get all the object properties that an individual has by using the following method:

```
anna = individuals[0]
object_properties = structural_reasoner.ind_object_properties(anna)
```

In this example, *object\_properties* contains all the object properties that *anna* has, which in our case would only be *hasChild*. Now we can get the individuals of this object property for *anna*.

```
for op in object_properties:
    object_properties_values = structural_reasoner.object_property_values(anna, op)
    for individual in object_properties_values:
        print(individual)
```

In this example we iterated over the *object\_properties*, assuming that there are more than 1, and we use the reasoner to get the values for each object property *op* of the individual *anna*. The values are individuals which we store in the variable *object\_properties\_values* and are printed in the end. The method *object\_property\_values* requires as the first argument, an *OWLNamedIndividual* that is the subject of the object property values and the second argument an *OWLObjectProperty* whose values are to be retrieved for the specified individual.

**NOTE:** You can as well get all the data properties of an individual in the same way by using *ind\_data\_properties* instead of *ind\_object\_properties* and *data\_property\_values* instead of *object\_property\_values*. Keep in mind that *data\_property\_values* returns literal values (type of *OWLLiteral*).

In the same way as with classes, you can also get the sub object properties or equivalent object properties.

```

from owlapy.owl_property import OWLObjectProperty

hasChild = OWLObjectProperty(URI(namespace, "hasChild"))

equivalent_to_hasChild = structural_reasoner.equivalent_object_properties(hasChild)
hasChild_sub_properties = structural_reasoner.sub_object_properties(hasChild)

```

In case you want to get the domains and ranges of an object property use the following:

```

hasChild_domains = structural_reasoner.object_property_domains(hasChild)
hasChild_ranges = structural_reasoner.object_property_ranges(hasChild)

```

**NOTE:** Again, you can do the same for data properties but instead of the word ‘object’ in the method name you should use ‘data’.

## 4.4 Find Instances

The method `instances` is a very convenient method. It takes only 1 argument that is basically a class expression and returns all the individuals belonging to that class expression. In Owlapy we have implemented a Python class for each type of class expression. The argument is of type *OWLClassExpression*.

Let us now show a simple example by finding the instances of the class *male* and printing them:

```

male_individuals = structural_reasoner.instances(male)
for ind in male_individuals:
    print(ind)

```

In this guide we covered the main functionalities of the reasoners in Owlapy. In the next one, we speak about owlapi synchronization and how can make use of owlapi in owlapy.

## 5 Owlapi Synchronization

As mentioned earlier, *owlapy* is loosely based in *owlapi*<sup>16</sup>, a library for ontology modification in java.

We have created *OWLAPIMapper*, a mapping class that makes possible the conversion of the most important classes from *owlapy* to *owlapi* and vice-versa.

We are able to use owlapi via *Jpype*<sup>17</sup>, a python module that provides access to Java in python. To start executing Java code via Jpype, one needs to start the java virtual machine (JVM). You don’t have to worry about it, because if a class is going to use *OWLAPIMapper* the JVM will start automatically. However, there is the function `startJVM` of the `static_functions.py` module if you ever need to start it manually.

### 5.1 “Sync” Classes

With the addition of the *OWLAPIMapper*, we introduce three new classes:

- *SyncOntologyManager*
- *SyncOntology*
- *SyncReasoner*

<sup>16</sup> <https://github.com/owlcs/owlapi>

<sup>17</sup> <https://jpype.readthedocs.io/en/latest/>

All the logic of these three classes is handled by *owlapi* through the mapper. They inherit from abstract classes already present in owlapy (OWLontologyManager, OWLontology and OWLReasoner respectively) so the usage is the same as other implementors of these abstract classes. However, there are also some extra methods, like `infer_axioms` of `SyncReasoner` which infers the missing axioms from the given ontology and returns them as `Iterable[OWLAxiom]`. Make sure to check the API docs to see them all.

To make this guide self-contained, we will go through a simple example showing how to use this above-mentioned classes:

```
from owlapy.owl_ontology_manager import SyncOntologyManager
from owlapy.owl_axiom import OWLDeclarationAxiom
from owlapy.class_expression import OWLClass
from owlapy.owl_reasoner import SyncReasoner
from owlapy.static_funcs import stopJVM

# (.) Create a manager and load the 'father' ontology
manager = SyncOntologyManager()
ontology = manager.load_ontology("KGs/Family/father.owl")

# (.) Use your ontology as you usually do
# (..) Add a new class
ontology.add_axiom(OWLDeclarationAxiom(OWLClass("http://example.com/father#some_new_
↪class"))))
# (..) Print classes in signature
[print(cls) for cls in ontology.classes_in_signature()]

# (.) Create a reasoner and perform some reasoning
reasoner = SyncReasoner(ontology)

# (..) Check ontology consistency
print(f"Is the ontology consistent? Answer: {reasoner.has_consistent_ontology()}")

# (..) Print all male individuals
[print(ind) for ind in reasoner.instances(OWLClass("http://example.com/father#male"))]

# (.) Stop the JVM if you no longer intend to use "Sync" classes
stopJVM()
```

This was a simple example using the *'father'* ontology to show just a small part of what you can do with “Sync” classes.

Notice that after we are done using them we can stop the JVM by either using `jpyype.shutdownJVM()` or the static function from the `static_functions.py` module `stopJVM()`. This will free the resources used by JPyype and the java packages. Once you stop the JVM it cannot be restarted so make sure you do that when you are done with the owlapi related classes. Stopping the JVM is not strictly necessary. The resources will be freed once the execution is over, but in case your code is somehow longer and the “Sync” classes only make up a part of your execution then you can stop the JVM after it not being needed anymore.

## 5.2 Notes

An important thing to keep in mind is that when starting the JVM you are able to import and use java classes as you would do in python (thanks to Jpyype). That means that you can play around with owlapi code in python as long as your JVM is started. Isn't that awesome!

`SyncReasoner` uses `HermiT` reasoner by default. You can choose between: “HermiT”, “Pellet”, “JFact” and “Openllet”. Although no significant difference is noticed between these reasoners, they surely differentiate in specific scenarios. You

can check owlapi Wiki<sup>18</sup> for more references.

**owlapi version: 5.1.9**

## 5.3 Examples

You can see usage examples in the *examples*<sup>19</sup> folder.

Test cases<sup>20</sup> can also serve as an example, so you can check them out as well.

## 6 Further Resources

Currently, we are working on our manuscript describing our framework. If you want to attribute our library, please use our GitHub page<sup>21</sup> for reference.

### 6.1 More Inside the Project

Examples and test cases provide a good starting point to get to know the project better. Find them in the folders *examples*<sup>22</sup> and *tests*<sup>23</sup>.

### 6.2 Contribution

Feel free to create a pull request and we will take a look on it. Your commitment is well appreciated!

### 6.3 Questions

In case you have any question, please contact: caglardemir8@gmail.com or open an issue on our GitHub issues page<sup>24</sup>.

### 6.4 Coverage Report

The coverage report is generated using coverage.py<sup>25</sup>.

Name	Stmts	Miss	Cover	Missing
owlapy/__init__.py	6	0	100%	
owlapy/abstracts/__init__.py	3	0	100%	
owlapy/abstracts/abstract_owl_ontology.py	16	1	94%	151
owlapy/abstracts/abstract_owl_ontology_manager.py	10	1	90%	24
owlapy/abstracts/abstract_owl_reasoner.py	49	10	80%	409-417, ↵
↪ 439, 464				
owlapy/class_expression/__init__.py	9	0	100%	
owlapy/class_expression/class_expression.py	34	3	91%	58, 62, 103
owlapy/class_expression/nary_boolean_expression.py	25	0	100%	
owlapy/class_expression/owl_class.py	33	1	97%	44
owlapy/class_expression/restriction.py	313	27	91%	41, 49, 68,

(continues on next page)

<sup>18</sup> <https://github.com/owlcs/owlapi/wiki>

<sup>19</sup> <https://github.com/dice-group/owlapy/tree/develop/examples>

<sup>20</sup> <https://github.com/dice-group/owlapy/tree/develop/tests>

<sup>21</sup> <https://github.com/dice-group/owlapy>

<sup>22</sup> <https://github.com/dice-group/owlapy/tree/develop/examples>

<sup>23</sup> <https://github.com/dice-group/owlapy/tree/develop/tests>

<sup>24</sup> <https://github.com/dice-group/owlapy/issues>

<sup>25</sup> <https://coverage.readthedocs.io/en/7.6.1/>

(continued from previous page)

↪ 71, 89, 171, 245-246, 303, 336, 342, 345, 419, 430, 439, 456, 502, 505, 582-583, ↪				
↪ 620, 663, 666, 706, 709, 758, 830				
owlapy/converter.py	420	174	59%	52-68, 75-
↪ 76, 79, 82, 152, 157, 169, 176, 184, 277, 294, 304-307, 313-359, 366-387, 394-401, ↪				
↪ 417-420, 431, 451, 460-481, 489-491, 498-511, 515-521, 525-548, 552-555, 559-560, ↪				
↪ 564-576, 580-587, 591-592, 621, 625-629				
owlapy/iri.py	79	6	92%	54, 69, 82,
↪ 97, 128, 133				
owlapy/meta_classes.py	11	0	100%	
owlapy/namespaces.py	27	3	89%	36, 40, 43
owlapy/owl_annotation.py	16	4	75%	16, 24, 42,
↪ 50				
owlapy/owl_axiom.py	519	130	75%	39, 42, 45,
↪ 59, 111-113, 116, 136-138, 141, 144, 147-150, 153, 182-184, 187, 190, 193, 196-200,				
↪ 203, 253-256, 261, 288, 291, 294, 332-335, 338-340, 343, 398-401, 404-406, 409, ↪				
↪ 536, 561-563, 566, 569, 572, 575, 578-581, 584, 623, 645-648, 652, 656, 674-675, ↪				
↪ 683, 692, 695-697, 700, 711, 734-738, 746, 754, 762, 765-767, 770, 787-789, 792, ↪				
↪ 795, 798-801, 804, 823-825, 828, 831, 834-837, 840, 859-861, 864, 867, 870-873, 876,				
↪ 909, 986, 1019, 1045, 1074, 1077, 1092, 1104, 1117, 1130, 1173, 1186-1188, 1191, ↪				
↪ 1209				
owlapy/owl_data_ranges.py	40	1	98%	46
owlapy/owl_datatype.py	20	2	90%	33-34
owlapy/owl_individual.py	20	1	95%	37
owlapy/owl_literal.py	286	66	77%	49, 77, 86,
↪ 90, 99, 103, 112, 116, 125, 129, 138, 142, 151, 155, 164, 169, 173, 203, 208, 217, ↪				
↪ 221, 244, 247-249, 258, 262, 288, 293, 302, 306, 311, 323, 329, 332-334, 337, 340, ↪				
↪ 346, 350, 355, 373, 378, 387, 391, 415, 420, 429, 433, 454, 459, 462-464, 467, 473, ↪				
↪ 477, 489-491, 494, 497-499, 502				
owlapy/owl_object.py	29	4	86%	26, 81-83
owlapy/owl_ontology.py	1000	222	78%	99, 110-
↪ 113, 116, 124, 142-148, 171, 179-182, 283-289, 312-321, 326-347, 367, 437, 440, 445-				
↪ 467, 472-482, 492-498, 510, 513-514, 554, 559-564, 574, 579, 596, 605-616, 621-636, ↪				
↪ 647, 652, 662, 674, 678, 714, 720, 731, 737, 742-766, 771-778, 807, 822-823, 841-				
↪ 844, 853, 861, 878, 890, 894, 907, 920, 928-929, 936-937, 942, 951-956, 963, 966-				
↪ 968, 971, 988, 992-993, 1017, 1020, 1023, 1026, 1029, 1036, 1074, 1083-1086, 1089-				
↪ 1092, 1097, 1100, 1140, 1150, 1166-1167, 1190-1191, 1270-1271, 1312, 1316, 1320, ↪				
↪ 1346, 1453, 1459, 1467				
owlapy/owl_ontology_manager.py	66	19	71%	26, 37, 41,
↪ 54-55, 63, 82, 85-89, 100-103, 112, 129-133				
owlapy/owl_property.py	69	11	84%	17, 24, 32,
↪ 40, 67, 76, 126, 158, 162, 174, 193				
owlapy/owl_reasoner.py	845	131	84%	170, 182-
↪ 184, 189-195, 202, 251-257, 263-265, 308-315, 341, 376-380, 406-409, 437-439, 441-				
↪ 443, 452, 465-467, 469-471, 478, 483-485, 505, 509-510, 523-525, 546, 591-593, 607-				
↪ 609, 627-628, 639-642, 645, 651, 675-684, 696, 701, 705, 753-756, 861-865, 887, 894,				
↪ 904-908, 916-920, 961-967, 979, 1100-1102, 1202, 1334, 1349, 1364, 1506-1527, 1558,				
↪ 1590				
owlapy/owlapi_mapper.py	357	62	83%	30, 106, ↪
↪ 125-126, 193-200, 206-209, 218, 222, 229, 236, 240, 244, 248, 252, 256, 260, 264, ↪				
↪ 268, 272, 276, 280, 285, 291, 295, 299, 312, 328, 340, 350, 361, 366, 376, 381, 398,				
↪ 416, 431, 436, 442, 447, 451, 456, 461, 466, 471-479, 508-509				
owlapy/parser.py	371	12	97%	316, 327, ↪

(continues on next page)

(continued from previous page)

↪400-401, 416, 656, 667, 751-752, 763, 779-780				
owlapy/providers.py	38	3	92%	41, 54, 56
owlapy/render.py	289	43	85%	78-113, ↪
↪142-157, 175, 179, 185, 221, 230, 235, 240, 374, 378, 420, 429, 434, 439				
owlapy/static_funcs.py	33	19	42%	21-26, 31-↪
↪42, 57-59				
owlapy/utils.py	794	304	62%	30-31, 54-↪
↪58, 75-86, 229, 233, 237, 243, 249-253, 257-261, 265, 269, 273, 279, 283, 287, 291, ↪				
↪295, 301, 307, 313, 317, 321, 325, 329-332, 336-339, 343, 350, 365-367, 370-379, ↪				
↪382, 385, 388, 391, 394, 398-404, 408, 419, 423, 427, 431, 435, 439-443, 447-451, ↪				
↪455-459, 463-467, 471, 475, 479-484, 488-493, 497-502, 506, 510, 514-518, 522-526, ↪				
↪530-534, 538-542, 546-550, 554, 558-562, 566, 570-575, 579-584, 588-593, 597, 601-↪				
↪605, 610, 619, 623, 627, 631, 635, 639, 643, 647-652, 656-662, 666, 670, 674, 679, ↪				
↪684, 689, 693, 697, 701, 705, 709-712, 716-719, 723, 727, 731, 736, 741, 746, 750, ↪				
↪799, 801, 803, 805, 807, 811, 813, 816, 818, 835, 861, 917, 931-933, 941-942, 962, ↪				
↪984-985, 1005, 1042-1043, 1058-1059, 1076-1104, 1109, 1114, 1123-1152, 1157, 1162-↪				
↪1164, 1229-1247, 1260-1262, 1267-1271				
owlapy/vocab.py	92	3	97%	32, 113-114
-----				
TOTAL	5919	1263	79%	

## 7 owlapy

### 7.1 Submodules

#### owlapy.abstracts

##### Submodules

#### owlapy.abstracts.abstract\_owl\_ontology

##### Classes

*AbstractOWLontology*

Represents an OWL 2 Ontology in the OWL 2 specification.

### Module Contents

**class** owlapy.abstracts.abstract\_owl\_ontology.**AbstractOWLontology**

Bases: *owlapy.owl\_object.OWLObject*

Represents an OWL 2 Ontology in the OWL 2 specification.

An OWLontology consists of a possibly empty set of OWLaxioms and a possibly empty set of OWLAnnotations. An ontology can have an ontology IRI which can be used to identify the ontology. If it has an ontology IRI then it may also have an ontology version IRI. Since OWL 2, an ontology need not have an ontology IRI. (See the OWL 2 Structural Specification).

An ontology cannot be modified directly. Changes must be applied via its OWLontologyManager.

**\_\_slots\_\_** = ()

**type\_index: Final = 1**

**abstract classes\_in\_signature ()** → Iterable[*owlapy.class\_expression.OWLClass*]

Gets the classes in the signature of this object.

**Returns**

Classes in the signature of this object.

**abstract data\_properties\_in\_signature ()** → Iterable[*owlapy.owl\_property.OWLDataProperty*]

Get the data properties that are in the signature of this object.

**Returns**

Data properties that are in the signature of this object.

**abstract object\_properties\_in\_signature ()**  
→ Iterable[*owlapy.owl\_property.OWLObjectProperty*]

A convenience method that obtains the object properties that are in the signature of this object.

**Returns**

Object properties that are in the signature of this object.

**abstract individuals\_in\_signature ()** → Iterable[*owlapy.owl\_individual.OWLNamedIndividual*]

A convenience method that obtains the individuals that are in the signature of this object.

**Returns**

Individuals that are in the signature of this object.

**abstract equivalent\_classes\_axioms (c: owlapy.class\_expression.OWLClass)**  
→ Iterable[*owlapy.owl\_axiom.OWLEquivalentClassesAxiom*]

Gets all of the equivalent axioms in this ontology that contain the specified class as an operand.

**Parameters**

**c** – The class for which the EquivalentClasses axioms should be retrieved.

**Returns**

EquivalentClasses axioms contained in this ontology.

**abstract general\_class\_axioms ()** → Iterable[*owlapy.owl\_axiom.OWLClassAxiom*]

**Get the general class axioms of this ontology. This includes SubClass axioms with a complex class expression**

as the sub class and EquivalentClass axioms and DisjointClass axioms with only complex class expressions.

**Returns**

General class axioms contained in this ontology.

**abstract data\_property\_domain\_axioms (property: owlapy.owl\_property.OWLDataProperty)**  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyDomainAxiom*]

Gets the OWLDataPropertyDomainAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**abstract data\_property\_range\_axioms (property: owlapy.owl\_property.OWLDataProperty)**  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyRangeAxiom*]

Gets the OWLDataPropertyRangeAxiom objects where the property is equal to the specified property.



**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

```
abstract object_property_domain_axioms (property: owlapy.owl_property.OWLObjectProperty)
    → Iterable[owlapy.owl_axiom.OWLObjectPropertyDomainAxiom]
```

Gets the OWLObjectPropertyDomainAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

```
abstract object_property_range_axioms (property: owlapy.owl_property.OWLObjectProperty)
    → Iterable[owlapy.owl_axiom.OWLObjectPropertyRangeAxiom]
```

Gets the OWLObjectPropertyRangeAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

```
abstract get_owl_ontology_manager () → _M
```

Gets the manager that manages this ontology.

```
abstract get_ontology_id () → _OI
```

Gets the OWLOntologyID belonging to this object.

**Returns**

The OWLOntologyID.

```
is_anonymous () → bool
```

Check whether this ontology does contain an IRI or not.

```
abstract add_axiom (
    axiom: owlapy.owl_axiom.OWLAxiom | Iterable[owlapy.owl_axiom.OWLAxiom])
```

Add the specified axiom/axioms to the ontology.

**Parameters**

**axiom** – Can be a single axiom or a collection of axioms.

**Returns**

Nothing.

```
abstract remove_axiom (
    axiom: owlapy.owl_axiom.OWLAxiom | Iterable[owlapy.owl_axiom.OWLAxiom])
```

Removes the specified axiom/axioms to the ontology.

**Parameters**

**axiom** – Can be a single axiom or a collection of axioms.

**Returns**

Nothing.

```
abstract save (document_iri: owlapy.iri.IRI | None = None)
```

Saves this ontology, using its IRI to determine where/how the ontology should be saved.

### Parameters

**document\_iri** – Whether you want to save in a different location.

## owlapy.abstracts.abstract\_owl\_ontology\_manager

### Classes

<i>AbstractOWLontologyChange</i>	Represents an ontology change.
<i>AbstractOWLontologyManager</i>	An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing

### Module Contents

```
class owlapy.abstracts.abstract_owl_ontology_manager.AbstractOWLontologyChange(  
    ontology: owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology)
```

Represents an ontology change.

```
__slots__ = ()
```

```
get_ontology() → owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology
```

Gets the ontology that the change is/was applied to.

#### Returns

The ontology that the change is applicable to.

```
class owlapy.abstracts.abstract_owl_ontology_manager.AbstractOWLontologyManager
```

An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing ontologies.

```
abstract create_ontology(iri: str | owlapy.iri.IRI)  
    → owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology
```

Creates a new (empty) ontology that that has the specified ontology IRI (and no version IRI).

#### Parameters

**iri** – The IRI of the ontology to be created, can also be a string.

#### Returns

The newly created ontology.

```
abstract load_ontology(iri: owlapy.iri.IRI | str)  
    → owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology
```

Loads an ontology that is assumed to have the specified ontology IRI as its IRI or version IRI. The ontology IRI will be mapped to an ontology document IRI.

#### Parameters

**iri** –

**The IRI that identifies the ontology, can also be a string.**

It is expected that the ontology will also have this IRI

(although the OWL API should tolerate situations where this is not the case).

#### Returns

The OWLOntology representation of the ontology that was loaded.

**abstract apply\_change** (*change: AbstractOWLontologyChange*)

A convenience method that applies just one change to an ontology. When this method is used through an OWLOntologyManager implementation, the instance used should be the one that the ontology returns through the get\_owl\_ontology\_manager() call.

**Parameters**

**change** – The change to be applied.

**Raises**

**ChangeApplied.UNSUCCESSFULLY** – if the change was not applied successfully.

## owlapy.abstracts.abstract\_owl\_reasoner

OWL Reasoner

### Attributes

*logger*

### Classes

*AbstractOWLReasoner*

An OWLReasoner reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of

### Module Contents

owlapy.abstracts.abstract\_owl\_reasoner.**logger**

**class** owlapy.abstracts.abstract\_owl\_reasoner.**AbstractOWLReasoner** (  
*ontology: owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLontology*)

An OWLReasoner reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of a particular ontology - the “root” ontology.

**\_\_slots\_\_** = ()

**abstract data\_property\_domains** (*pe: owlapy.owl\_property.OWLDataProperty*,  
*direct: bool = False*) → Iterable[*owlapy.class\_expression.OWLClassExpression*]

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

**Parameters**

- **pe** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

**Returns**

Let N = equivalent\_classes(DataSomeValuesFrom(pe rdfs:Literal)). If direct is True: then if N is not empty then the return value is N, else the return value is the result of super\_classes(DataSomeValuesFrom(pe rdfs:Literal), true). If direct is False: then the result of

`super_classes(DataSomeValuesFrom(pe rdfs:Literal), false)` together with N if N is non-empty.  
(Note, `rdfs:Literal` is the top datatype).

**abstract** `object_property_domains` (*pe*: `owlapy.owl_property.OWLObjectProperty`,  
*direct*: `bool = False`)  $\rightarrow$  `Iterable[owlapy.class_expression.OWLClassExpression]`

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **pe** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

#### Returns

Let N = `equivalent_classes(ObjectSomeValuesFrom(pe owl:Thing))`. If `direct` is True: then if N is not empty then the return value is N, else the return value is the result of `super_classes(ObjectSomeValuesFrom(pe owl:Thing), true)`. If `direct` is False: then the result of `super_classes(ObjectSomeValuesFrom(pe owl:Thing), false)` together with N if N is non-empty.

**abstract** `object_property_ranges` (*pe*: `owlapy.owl_property.OWLObjectProperty`,  
*direct*: `bool = False`)  $\rightarrow$  `Iterable[owlapy.class_expression.OWLClassExpression]`

**Gets the class expressions that are the direct or indirect ranges of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **pe** – The property expression whose ranges are to be retrieved.
- **direct** – Specifies if the direct ranges should be retrieved (True), or if all ranges should be retrieved (False).

#### Returns

Let N = `equivalent_classes(ObjectSomeValuesFrom(ObjectInverseOf(pe) owl:Thing))`. If `direct` is True: then if N is not empty then the return value is N, else the return value is the result of `super_classes(ObjectSomeValuesFrom(ObjectInverseOf(pe) owl:Thing), true)`. If `direct` is False: then the result of `super_classes(ObjectSomeValuesFrom(ObjectInverseOf(pe) owl:Thing), false)` together with N if N is non-empty.

**abstract** `equivalent_classes` (*ce*: `owlapy.class_expression.OWLClassExpression`)  
 $\rightarrow$  `Iterable[owlapy.class_expression.OWLClassExpression]`

**Gets the class expressions that are equivalent to the specified class expression with respect to the set of reasoner axioms.**

#### Parameters

**ce** – The class expression whose equivalent classes are to be retrieved.

#### Returns

All class expressions C where the root ontology imports closure entails `EquivalentClasses(ce C)`. If `ce` is not a class name (i.e. it is an anonymous class expression) and there are no such classes C then there will be no result. If `ce` is unsatisfiable with respect to the set of reasoner axioms then `owl:Nothing`, i.e. the bottom node, will be returned.

**abstract disjoint\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*)  
→ *Iterable[owlapy.class\_expression.OWLClassExpression]*

Gets the class expressions that are disjoint with specified class expression with respect to the set of reasoner axioms.

**Parameters**

**ce** – The class expression whose disjoint classes are to be retrieved.

**Returns**

All class expressions D where the set of reasoner axioms entails *EquivalentClasses(D ObjectComplementOf(ce))* or *StrictSubClassOf(D ObjectComplementOf(ce))*.

**abstract different\_individuals** (*ind*: *owlapy.owl\_individual.OWLNamedIndividual*)  
→ *Iterable[owlapy.owl\_individual.OWLNamedIndividual]*

Gets the individuals that are different from the specified individual with respect to the set of reasoner axioms.

**Parameters**

**ind** – The individual whose different individuals are to be retrieved.

**Returns**

All individuals x where the set of reasoner axioms entails *DifferentIndividuals(ind x)*.

**abstract same\_individuals** (*ind*: *owlapy.owl\_individual.OWLNamedIndividual*)  
→ *Iterable[owlapy.owl\_individual.OWLNamedIndividual]*

Gets the individuals that are the same as the specified individual with respect to the set of reasoner axioms.

**Parameters**

**ind** – The individual whose same individuals are to be retrieved.

**Returns**

All individuals x where the root ontology imports closure entails *SameIndividual(ind x)*.

**abstract equivalent\_object\_properties** (  
*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*)  
→ *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the simplified object properties that are equivalent to the specified object property with respect to the set of reasoner axioms.

**Parameters**

**op** – The object property whose equivalent object properties are to be retrieved.

**Returns**

All simplified object properties e where the root ontology imports closure entails *EquivalentObjectProperties(op e)*. If op is unsatisfiable with respect to the set of reasoner axioms then *owl:bottomDataProperty* will be returned.

**abstract equivalent\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*)  
→ *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the data properties that are equivalent to the specified data property with respect to the set of reasoner axioms.

**Parameters**

**dp** – The data property whose equivalent data properties are to be retrieved.

**Returns**

All data properties e where the root ontology imports closure entails *EquivalentDataProperties(dp e)*. If dp is unsatisfiable with respect to the set of reasoner axioms then *owl:bottomDataProperty* will be returned.

**abstract data\_property\_values** (*e*: owlapy.owl\_object.OWLEntity,  
*pe*: owlapy.owl\_property.OWLDataProperty) → Iterable[owlapy.owl\_literal.OWLLiteral]

Gets the data property values for the specified entity and data property expression.

#### Parameters

- **e** – The owl entity (usually an individual) that is the subject of the data property values.
- **pe** – The data property expression whose values are to be retrieved for the specified entity.

Note: Can be used to get values, for example, of ‘label’ property of owl entities such as classes and properties too (not only individuals).

#### Returns

A set of OWLLiterals containing literals such that for each literal *l* in the set, the set of reasoner axioms entails DataPropertyAssertion(*pe* ind *l*).

**abstract object\_property\_values** (*ind*: owlapy.owl\_individual.OWLNamedIndividual,  
*pe*: owlapy.owl\_property.OWLObjectPropertyExpression)  
→ Iterable[owlapy.owl\_individual.OWLNamedIndividual]

Gets the object property values for the specified individual and object property expression.

#### Parameters

- **ind** – The individual that is the subject of the object property values.
- **pe** – The object property expression whose values are to be retrieved for the specified individual.

#### Returns

The named individuals such that for each individual *j*, the set of reasoner axioms entails ObjectPropertyAssertion(*pe* ind *j*).

**abstract instances** (*ce*: owlapy.class\_expression.OWLClassExpression, *direct*: bool = False,  
*timeout*: int = 1000) → Iterable[owlapy.owl\_individual.OWLNamedIndividual]

Gets the individuals which are instances of the specified class expression.

#### Parameters

- **ce** – The class expression whose instances are to be retrieved.
- **direct** – Specifies if the direct instances should be retrieved (True), or if all instances should be retrieved (False).
- **timeout** – Time limit in seconds until results must be returned, else empty set is returned.

#### Returns

If *direct* is True, each named individual *j* where the set of reasoner axioms entails DirectClassAssertion(*ce*, *j*). If *direct* is False, each named individual *j* where the set of reasoner axioms entails ClassAssertion(*ce*, *j*). If *ce* is unsatisfiable with respect to the set of reasoner axioms then nothing returned.

**abstract sub\_classes** (*ce*: owlapy.class\_expression.OWLClassExpression, *direct*: bool = False)  
→ Iterable[owlapy.class\_expression.OWLClassExpression]

Gets the set of named classes that are the strict (potentially direct) subclasses of the specified class expression with respect to the reasoner axioms.

#### Parameters

- **ce** – The class expression whose strict (direct) subclasses are to be retrieved.
- **direct** – Specifies if the direct subclasses should be retrieved (True) or if the all subclasses (descendant) classes should be retrieved (False).

### Returns

If direct is True, each class C where reasoner axioms entails `DirectSubClassOf(C, ce)`. If direct is False, each class C where reasoner axioms entails `StrictSubClassOf(C, ce)`. If ce is equivalent to `owl:Nothing` then nothing will be returned.

**abstract disjoint\_object\_properties** (*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*)  
→ *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the simplified object properties that are disjoint with the specified object property with respect to the set of reasoner axioms.

### Parameters

**op** – The object property whose disjoint object properties are to be retrieved.

### Returns

All simplified object properties e where the root ontology imports closure entails `EquivalentObjectProperties(e ObjectPropertyComplementOf(op))` or `StrictSubObjectPropertyOf(e ObjectPropertyComplementOf(op))`.

**abstract disjoint\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*)  
→ *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the data properties that are disjoint with the specified data property with respect to the set of reasoner axioms.

### Parameters

**dp** – The data property whose disjoint data properties are to be retrieved.

### Returns

All data properties e where the root ontology imports closure entails `EquivalentDataProperties(e DataPropertyComplementOf(dp))` or `StrictSubDataPropertyOf(e DataPropertyComplementOf(dp))`.

**abstract sub\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*, *direct*: *bool = False*)  
→ *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the set of named data properties that are the strict (potentially direct) subproperties of the specified data property expression with respect to the imports closure of the root ontology.

### Parameters

- **dp** – The data property whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

### Returns

If direct is True, each property P where the set of reasoner axioms entails `DirectSubDataPropertyOf(P, pe)`. If direct is False, each property P where the set of reasoner axioms entails `StrictSubDataPropertyOf(P, pe)`. If pe is equivalent to `owl:bottomDataProperty` then nothing will be returned.

**abstract super\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*,  
*direct*: *bool = False*) → *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the stream of data properties that are the strict (potentially direct) super properties of the specified data property with respect to the imports closure of the root ontology.

### Parameters

- **dp** (*OWLDataProperty*) – The data property whose super properties are to be retrieved.
- **direct** (*bool*) – Specifies if the direct super properties should be retrieved (True) or if the all super properties (ancestors) should be retrieved (False).

### Returns

Iterable of super properties.

**abstract sub\_object\_properties** (*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*,  
*direct*: *bool* = *False*) → *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the stream of simplified object property expressions that are the strict (potentially direct) subproperties of the specified object property expression with respect to the imports closure of the root ontology.

### Parameters

- **op** – The object property expression whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

### Returns

If **direct** is True, simplified object property expressions, such that for each simplified object property expression, P, the set of reasoner axioms entails *DirectSubObjectPropertyOf*(P, pe). If **direct** is False, simplified object property expressions, such that for each simplified object property expression, P, the set of reasoner axioms entails *StrictSubObjectPropertyOf*(P, pe). If pe is equivalent to *owl:bottomObjectProperty* then nothing will be returned.

**abstract super\_object\_properties** (*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*,  
*direct*: *bool* = *False*) → *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the stream of object properties that are the strict (potentially direct) super properties of the specified object property with respect to the imports closure of the root ontology.

### Parameters

- **op** (*OWLObjectPropertyExpression*) – The object property expression whose super properties are to be retrieved.
- **direct** (*bool*) – Specifies if the direct super properties should be retrieved (True) or if the all super properties (ancestors) should be retrieved (False).

### Returns

Iterable of super properties.

**abstract types** (*ind*: *owlapy.owl\_individual.OWLNamedIndividual*, *direct*: *bool* = *False*)  
→ *Iterable[owlapy.class\_expression.OWLClass]*

Gets the named classes which are (potentially direct) types of the specified named individual.

### Parameters

- **ind** – The individual whose types are to be retrieved.
- **direct** – Specifies if the direct types should be retrieved (True), or if all types should be retrieved (False).

### Returns

If **direct** is True, each named class C where the set of reasoner axioms entails *DirectClassAssertion*(C, ind). If **direct** is False, each named class C where the set of reasoner axioms entails *ClassAssertion*(C, ind).

**abstract get\_root\_ontology** () → *owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLontology*

Gets the “root” ontology that is loaded into this reasoner. The reasoner takes into account the axioms in this ontology and its import’s closure.

**abstract super\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *direct*: *bool* = *False*)  
→ *Iterable[owlapy.class\_expression.OWLClassExpression]*

Gets the stream of named classes that are the strict (potentially direct) super classes of the specified class expression with respect to the imports closure of the root ontology.



### Parameters

- **ce** – The class expression whose strict (direct) super classes are to be retrieved.
- **direct** – Specifies if the direct super classes should be retrieved (True) or if the all super classes (ancestors) classes should be retrieved (False).

### Returns

If direct is True, each class C where the set of reasoner axioms entails `DirectSubClassOf(ce, C)`. If direct is False, each class C where set of reasoner axioms entails `StrictSubClassOf(ce, C)`. If ce is equivalent to `owl:Thing` then nothing will be returned.

**data\_property\_ranges** (*pe: owlapy.owl\_property.OWLDataProperty, direct: bool = False*)  
→ `Iterable[owlapy.owl_data_ranges.OWLDataRange]`

Gets the data ranges that are the direct or indirect ranges of this property with respect to the imports closure of the root ontology.

### Parameters

- **pe** – The property expression whose ranges are to be retrieved.
- **direct** – Specifies if the direct ranges should be retrieved (True), or if all ranges should be retrieved (False).

Returns:

**all\_data\_property\_values** (*pe: owlapy.owl\_property.OWLDataProperty, direct: bool = True*)  
→ `Iterable[owlapy.owl_literal.OWLLiteral]`

Gets all values for the given data property expression that appear in the knowledge base.

### Parameters

- **pe** – The data property expression whose values are to be retrieved
- **direct** – Specifies if only the direct values of the data property pe should be retrieved (True), or if the values of sub properties of pe should be taken into account (False).

### Returns

A set of OWLLiterals containing literals such that for each literal l in the set, the set of reasoner axioms entails `DataPropertyAssertion(pe ind l)` for any ind.

**ind\_data\_properties** (*ind: owlapy.owl\_individual.OWLNamedIndividual, direct: bool = True*)  
→ `Iterable[owlapy.owl_property.OWLDataProperty]`

Gets all data properties for the given individual that appear in the knowledge base.

### Parameters

- **ind** – The named individual whose data properties are to be retrieved
- **direct** – Specifies if the direct data properties should be retrieved (True), or if all data properties should be retrieved (False), so that sub properties are taken into account.

### Returns

All data properties pe where the set of reasoner axioms entails `DataPropertyAssertion(pe ind l)` for atleast one l.

**ind\_object\_properties** (*ind: owlapy.owl\_individual.OWLNamedIndividual, direct: bool = True*)  
→ `Iterable[owlapy.owl_property.OWLObjectProperty]`

Gets all object properties for the given individual that appear in the knowledge base.

### Parameters

- **ind** – The named individual whose object properties are to be retrieved

- **direct** – Specifies if the direct object properties should be retrieved (True), or if all object properties should be retrieved (False), so that sub properties are taken into account.

#### Returns

All data properties *pe* where the set of reasoner axioms entails `ObjectPropertyAssertion(pe ind ind2)` for atleast one *ind2*.

## Classes

<i>AbstractOWLOntologyManager</i>	An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing
<i>AbstractOWLOntologyChange</i>	Represents an ontology change.
<i>AbstractOWLOntology</i>	Represents an OWL 2 Ontology in the OWL 2 specification.
<i>AbstractOWLReasoner</i>	An OWLReasoner reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of

## Package Contents

**class** owlapy.abstracts.**AbstractOWLOntologyManager**

An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing ontologies.

**abstract** `create_ontology(iri: str | owlapy.iri.IRI)`  
→ `owlapy.abstracts.abstract_owl_ontology.AbstractOWLOntology`

Creates a new (empty) ontology that has the specified ontology IRI (and no version IRI).

#### Parameters

**iri** – The IRI of the ontology to be created, can also be a string.

#### Returns

The newly created ontology.

**abstract** `load_ontology(iri: owlapy.iri.IRI | str)`  
→ `owlapy.abstracts.abstract_owl_ontology.AbstractOWLOntology`

Loads an ontology that is assumed to have the specified ontology IRI as its IRI or version IRI. The ontology IRI will be mapped to an ontology document IRI.

#### Parameters

**iri** –

**The IRI that identifies the ontology, can also be a string.**

It is expected that the ontology will also have this IRI

(although the OWL API should tolerate situations where this is not the case).

#### Returns

The OWLOntology representation of the ontology that was loaded.

**abstract** `apply_change(change: AbstractOWLOntologyChange)`

A convenience method that applies just one change to an ontology. When this method is used through an OWLOntologyManager implementation, the instance used should be the one that the ontology returns through the `get_owl_ontology_manager()` call.

#### Parameters

**change** – The change to be applied.

### Raises

**ChangeApplied.UNSUCCESSFULLY** – if the change was not applied successfully.

```
class owlapy.abstracts.AbstractOWLontologyChange (
    ontology: owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology)
    Represents an ontology change.

    __slots__ = ()

    get_ontology() → owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology
        Gets the ontology that the change is/was applied to.
```

### Returns

The ontology that the change is applicable to.

```
class owlapy.abstracts.AbstractOWLontology
    Bases: owlapy.owl_object.OWLObject

    Represents an OWL 2 Ontology in the OWL 2 specification.

    An OWLontology consists of a possibly empty set of OWLAxioms and a possibly empty set of OWLAnnotations.
    An ontology can have an ontology IRI which can be used to identify the ontology. If it has an ontology IRI then it
    may also have an ontology version IRI. Since OWL 2, an ontology need not have an ontology IRI. (See the OWL
    2 Structural Specification).
```

An ontology cannot be modified directly. Changes must be applied via its OWLontologyManager.

```
__slots__ = ()

type_index: Final = 1

abstract classes_in_signature() → Iterable[owlapy.class_expression.OWLClass]
    Gets the classes in the signature of this object.
```

### Returns

Classes in the signature of this object.

```
abstract data_properties_in_signature() → Iterable[owlapy.owl_property.OWLDataProperty]
    Get the data properties that are in the signature of this object.
```

### Returns

Data properties that are in the signature of this object.

```
abstract object_properties_in_signature()
    → Iterable[owlapy.owl_property.OWLObjectProperty]
    A convenience method that obtains the object properties that are in the signature of this object.
```

### Returns

Object properties that are in the signature of this object.

```
abstract individuals_in_signature() → Iterable[owlapy.owl_individual.OWLNamedIndividual]
    A convenience method that obtains the individuals that are in the signature of this object.
```

### Returns

Individuals that are in the signature of this object.

```
abstract equivalent_classes_axioms(c: owlapy.class_expression.OWLClass)
    → Iterable[owlapy.owl_axiom.OWLEquivalentClassesAxiom]
    Gets all of the equivalent axioms in this ontology that contain the specified class as an operand.
```

### Parameters

**c** – The class for which the EquivalentClasses axioms should be retrieved.

### Returns

EquivalentClasses axioms contained in this ontology.

**abstract general\_class\_axioms ()** → Iterable[*owlapy.owl\_axiom.OWLClassAxiom*]

**Get the general class axioms of this ontology. This includes SubClass axioms with a complex class expression**

as the sub class and EquivalentClass axioms and DisjointClass axioms with only complex class expressions.

### Returns

General class axioms contained in this ontology.

**abstract data\_property\_domain\_axioms** (*property: owlapy.owl\_property.OWLDataProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyDomainAxiom*]

Gets the OWLDataPropertyDomainAxiom objects where the property is equal to the specified property.

### Parameters

**property** – The property which is equal to the property of the retrieved axioms.

### Returns

The axioms matching the search.

**abstract data\_property\_range\_axioms** (*property: owlapy.owl\_property.OWLDataProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyRangeAxiom*]

Gets the OWLDataPropertyRangeAxiom objects where the property is equal to the specified property.

### Parameters

**property** – The property which is equal to the property of the retrieved axioms.

### Returns

The axioms matching the search.

**abstract object\_property\_domain\_axioms** (*property: owlapy.owl\_property.OWLObjectProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLObjectPropertyDomainAxiom*]

Gets the OWLObjectPropertyDomainAxiom objects where the property is equal to the specified property.

### Parameters

**property** – The property which is equal to the property of the retrieved axioms.

### Returns

The axioms matching the search.

**abstract object\_property\_range\_axioms** (*property: owlapy.owl\_property.OWLObjectProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLObjectPropertyRangeAxiom*]

Gets the OWLObjectPropertyRangeAxiom objects where the property is equal to the specified property.

### Parameters

**property** – The property which is equal to the property of the retrieved axioms.

### Returns

The axioms matching the search.

**abstract get\_owl\_ontology\_manager ()** → *\_M*

Gets the manager that manages this ontology.

**abstract get\_ontology\_id ()** → *\_OI*

Gets the OWLOntologyID belonging to this object.

### Returns

The OWLOntologyID.

**is\_anonymous** () → bool

Check whether this ontology does contain an IRI or not.

**abstract add\_axiom** (  
    *axiom: owlapy.owl\_axiom.OWLXiom | Iterable[owlapy.owl\_axiom.OWLXiom]*)

Add the specified axiom/axioms to the ontology.

**Parameters**

**axiom** – Can be a single axiom or a collection of axioms.

**Returns**

Nothing.

**abstract remove\_axiom** (  
    *axiom: owlapy.owl\_axiom.OWLXiom | Iterable[owlapy.owl\_axiom.OWLXiom]*)

Removes the specified axiom/axioms to the ontology.

**Parameters**

**axiom** – Can be a single axiom or a collection of axioms.

**Returns**

Nothing.

**abstract save** (*document\_iri: owlapy.iri.IRI | None = None*)

Saves this ontology, using its IRI to determine where/how the ontology should be saved.

**Parameters**

**document\_iri** – Whether you want to save in a different location.

**class owlapy.abstracts.AbstractOWLReasoner** (  
    *ontology: owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLontology*)

An OWLReasoner reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of a particular ontology - the “root” ontology.

**\_\_slots\_\_** = ()

**abstract data\_property\_domains** (*pe: owlapy.owl\_property.OWLDataProperty,*  
    *direct: bool = False*) → *Iterable[owlapy.class\_expression.OWLClassExpression]*

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

**Parameters**

- **pe** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

**Returns**

Let N = `equivalent_classes(DataSomeValuesFrom(pe rdfs:Literal))`. If `direct` is True: then if N is not empty then the return value is N, else the return value is the result of `super_classes(DataSomeValuesFrom(pe rdfs:Literal), true)`. If `direct` is False: then the result of `super_classes(DataSomeValuesFrom(pe rdfs:Literal), false)` together with N if N is non-empty. (Note, `rdfs:Literal` is the top datatype).

**abstract object\_property\_domains** (*pe: owlapy.owl\_property.OWLObjectProperty,*  
    *direct: bool = False*) → *Iterable[owlapy.class\_expression.OWLClassExpression]*

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **pe** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

#### Returns

Let  $N = \text{equivalent\_classes}(\text{ObjectSomeValuesFrom}(\text{pe owl:Thing}))$ . If **direct** is True: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{pe owl:Thing}), \text{true})$ . If **direct** is False: then the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{pe owl:Thing}), \text{false})$  together with  $N$  if  $N$  is non-empty.

```
abstract object_property_ranges (pe: owlapy.owl_property.OWLObjectProperty,  
direct: bool = False) → Iterable[owlapy.class_expression.OWLClassExpression]
```

**Gets the class expressions that are the direct or indirect ranges of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **pe** – The property expression whose ranges are to be retrieved.
- **direct** – Specifies if the direct ranges should be retrieved (True), or if all ranges should be retrieved (False).

#### Returns

Let  $N = \text{equivalent\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe owl:Thing})))$ . If **direct** is True: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe owl:Thing})), \text{true})$ . If **direct** is False: then the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe owl:Thing})), \text{false})$  together with  $N$  if  $N$  is non-empty.

```
abstract equivalent_classes (ce: owlapy.class_expression.OWLClassExpression)  
→ Iterable[owlapy.class_expression.OWLClassExpression]
```

**Gets the class expressions that are equivalent to the specified class expression with respect to the set of reasoner axioms.**

#### Parameters

**ce** – The class expression whose equivalent classes are to be retrieved.

#### Returns

All class expressions  $C$  where the root ontology imports closure entails  $\text{EquivalentClasses}(ce\ C)$ . If  $ce$  is not a class name (i.e. it is an anonymous class expression) and there are no such classes  $C$  then there will be no result. If  $ce$  is unsatisfiable with respect to the set of reasoner axioms then  $\text{owl:Nothing}$ , i.e. the bottom node, will be returned.

```
abstract disjoint_classes (ce: owlapy.class_expression.OWLClassExpression)  
→ Iterable[owlapy.class_expression.OWLClassExpression]
```

**Gets the class expressions that are disjoint with specified class expression with respect to the set of reasoner axioms.**

#### Parameters

**ce** – The class expression whose disjoint classes are to be retrieved.

### Returns

All class expressions D where the set of reasoner axioms entails `EquivalentClasses(D ObjectComplementOf(ce))` or `StrictSubClassOf(D ObjectComplementOf(ce))`.

**abstract different\_individuals** (*ind*: *owlapy.owl\_individual.OWLNamedIndividual*)  
→ *Iterable[owlapy.owl\_individual.OWLNamedIndividual]*

Gets the individuals that are different from the specified individual with respect to the set of reasoner axioms.

### Parameters

**ind** – The individual whose different individuals are to be retrieved.

### Returns

All individuals x where the set of reasoner axioms entails `DifferentIndividuals(ind x)`.

**abstract same\_individuals** (*ind*: *owlapy.owl\_individual.OWLNamedIndividual*)  
→ *Iterable[owlapy.owl\_individual.OWLNamedIndividual]*

Gets the individuals that are the same as the specified individual with respect to the set of reasoner axioms.

### Parameters

**ind** – The individual whose same individuals are to be retrieved.

### Returns

All individuals x where the root ontology imports closure entails `SameIndividual(ind x)`.

**abstract equivalent\_object\_properties** (*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*)  
→ *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the simplified object properties that are equivalent to the specified object property with respect to the set of reasoner axioms.

### Parameters

**op** – The object property whose equivalent object properties are to be retrieved.

### Returns

All simplified object properties e where the root ontology imports closure entails `EquivalentObjectProperties(op e)`. If op is unsatisfiable with respect to the set of reasoner axioms then `owl:bottomDataProperty` will be returned.

**abstract equivalent\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*)  
→ *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the data properties that are equivalent to the specified data property with respect to the set of reasoner axioms.

### Parameters

**dp** – The data property whose equivalent data properties are to be retrieved.

### Returns

All data properties e where the root ontology imports closure entails `EquivalentDataProperties(dp e)`. If dp is unsatisfiable with respect to the set of reasoner axioms then `owl:bottomDataProperty` will be returned.

**abstract data\_property\_values** (*e*: *owlapy.owl\_object.OWLEntity*,  
*pe*: *owlapy.owl\_property.OWLDataProperty*) → *Iterable[owlapy.owl\_literal.OWLLiteral]*

Gets the data property values for the specified entity and data property expression.

### Parameters

- **e** – The owl entity (usually an individual) that is the subject of the data property values.
- **pe** – The data property expression whose values are to be retrieved for the specified entity.

Note: Can be used to get values, for example, of 'label' property of owl entities such as classes and properties too (not only individuals).

#### Returns

A set of OWLLiterals containing literals such that for each literal *l* in the set, the set of reasoner axioms entails `DataPropertyAssertion(pe ind l)`.

```
abstract object_property_values (ind: owlapy.owl_individual.OWLNamedIndividual,  
    pe: owlapy.owl_property.OWLObjectPropertyExpression)  
    → Iterable[owlapy.owl_individual.OWLNamedIndividual]
```

Gets the object property values for the specified individual and object property expression.

#### Parameters

- **ind** – The individual that is the subject of the object property values.
- **pe** – The object property expression whose values are to be retrieved for the specified individual.

#### Returns

The named individuals such that for each individual *j*, the set of reasoner axioms entails `ObjectPropertyAssertion(pe ind j)`.

```
abstract instances (ce: owlapy.class_expression.OWLClassExpression, direct: bool = False,  
    timeout: int = 1000) → Iterable[owlapy.owl_individual.OWLNamedIndividual]
```

Gets the individuals which are instances of the specified class expression.

#### Parameters

- **ce** – The class expression whose instances are to be retrieved.
- **direct** – Specifies if the direct instances should be retrieved (True), or if all instances should be retrieved (False).
- **timeout** – Time limit in seconds until results must be returned, else empty set is returned.

#### Returns

If **direct** is True, each named individual *j* where the set of reasoner axioms entails `DirectClassAssertion(ce, j)`. If **direct** is False, each named individual *j* where the set of reasoner axioms entails `ClassAssertion(ce, j)`. If *ce* is unsatisfiable with respect to the set of reasoner axioms then nothing returned.

```
abstract sub_classes (ce: owlapy.class_expression.OWLClassExpression, direct: bool = False)  
    → Iterable[owlapy.class_expression.OWLClassExpression]
```

Gets the set of named classes that are the strict (potentially direct) subclasses of the specified class expression with respect to the reasoner axioms.

#### Parameters

- **ce** – The class expression whose strict (direct) subclasses are to be retrieved.
- **direct** – Specifies if the direct subclasses should be retrieved (True) or if the all subclasses (descendant) classes should be retrieved (False).

#### Returns

If **direct** is True, each class *C* where reasoner axioms entails `DirectSubClassOf(C, ce)`. If **direct** is False, each class *C* where reasoner axioms entails `StrictSubClassOf(C, ce)`. If *ce* is equivalent to `owl:Nothing` then nothing will be returned.

```
abstract disjoint_object_properties (op: owlapy.owl_property.OWLObjectPropertyExpression)  
    → Iterable[owlapy.owl_property.OWLObjectPropertyExpression]
```



Gets the simplified object properties that are disjoint with the specified object property with respect to the set of reasoner axioms.

**Parameters**

**op** – The object property whose disjoint object properties are to be retrieved.

**Returns**

All simplified object properties *e* where the root ontology imports closure entails `EquivalentObjectProperties(e ObjectPropertyComplementOf(op))` or `StrictSubObjectPropertyOf(e ObjectPropertyComplementOf(op))`.

```
abstract disjoint_data_properties (dp: owlapy.owl_property.OWLDataProperty)
    → Iterable[owlapy.owl_property.OWLDataProperty]
```

Gets the data properties that are disjoint with the specified data property with respect to the set of reasoner axioms.

**Parameters**

**dp** – The data property whose disjoint data properties are to be retrieved.

**Returns**

All data properties *e* where the root ontology imports closure entails `EquivalentDataProperties(e DataPropertyComplementOf(dp))` or `StrictSubDataPropertyOf(e DataPropertyComplementOf(dp))`.

```
abstract sub_data_properties (dp: owlapy.owl_property.OWLDataProperty, direct: bool = False)
    → Iterable[owlapy.owl_property.OWLDataProperty]
```

Gets the set of named data properties that are the strict (potentially direct) subproperties of the specified data property expression with respect to the imports closure of the root ontology.

**Parameters**

- **dp** – The data property whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

**Returns**

If **direct** is True, each property *P* where the set of reasoner axioms entails `DirectSubDataPropertyOf(P, pe)`. If **direct** is False, each property *P* where the set of reasoner axioms entails `StrictSubDataPropertyOf(P, pe)`. If *pe* is equivalent to `owl:bottomDataProperty` then nothing will be returned.

```
abstract super_data_properties (dp: owlapy.owl_property.OWLDataProperty,
    direct: bool = False) → Iterable[owlapy.owl_property.OWLDataProperty]
```

Gets the stream of data properties that are the strict (potentially direct) super properties of the specified data property with respect to the imports closure of the root ontology.

**Parameters**

- **dp** (`OWLDataProperty`) – The data property whose super properties are to be retrieved.
- **direct** (`bool`) – Specifies if the direct super properties should be retrieved (True) or if the all super properties (ancestors) should be retrieved (False).

**Returns**

Iterable of super properties.

```
abstract sub_object_properties (op: owlapy.owl_property.OWLObjectPropertyExpression,
    direct: bool = False) → Iterable[owlapy.owl_property.OWLObjectPropertyExpression]
```

Gets the stream of simplified object property expressions that are the strict (potentially direct) subproperties of the specified object property expression with respect to the imports closure of the root ontology.

#### Parameters

- **op** – The object property expression whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

#### Returns

If **direct** is True, simplified object property expressions, such that for each simplified object property expression, P, the set of reasoner axioms entails `DirectSubObjectPropertyOf(P, pe)`. If **direct** is False, simplified object property expressions, such that for each simplified object property expression, P, the set of reasoner axioms entails `StrictSubObjectPropertyOf(P, pe)`. If `pe` is equivalent to `owl:bottomObjectProperty` then nothing will be returned.

**abstract super\_object\_properties** (*op: owlapy.owl\_property.OWLObjectPropertyExpression, direct: bool = False*) → `Iterable[owlapy.owl_property.OWLObjectPropertyExpression]`

Gets the stream of object properties that are the strict (potentially direct) super properties of the specified object property with respect to the imports closure of the root ontology.

#### Parameters

- **op** (`OWLObjectPropertyExpression`) – The object property expression whose super properties are to be retrieved.
- **direct** (`bool`) – Specifies if the direct super properties should be retrieved (True) or if the all super properties (ancestors) should be retrieved (False).

#### Returns

Iterable of super properties.

**abstract types** (*ind: owlapy.owl\_individual.OWLNamedIndividual, direct: bool = False*) → `Iterable[owlapy.class_expression.OWLClass]`

Gets the named classes which are (potentially direct) types of the specified named individual.

#### Parameters

- **ind** – The individual whose types are to be retrieved.
- **direct** – Specifies if the direct types should be retrieved (True), or if all types should be retrieved (False).

#### Returns

If **direct** is True, each named class C where the set of reasoner axioms entails `DirectClassAssertion(C, ind)`. If **direct** is False, each named class C where the set of reasoner axioms entails `ClassAssertion(C, ind)`.

**abstract get\_root\_ontology** () → `owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology`

Gets the “root” ontology that is loaded into this reasoner. The reasoner takes into account the axioms in this ontology and its import’s closure.

**abstract super\_classes** (*ce: owlapy.class\_expression.OWLClassExpression, direct: bool = False*) → `Iterable[owlapy.class_expression.OWLClassExpression]`

Gets the stream of named classes that are the strict (potentially direct) super classes of the specified class expression with respect to the imports closure of the root ontology.

#### Parameters

- **ce** – The class expression whose strict (direct) super classes are to be retrieved.

- **direct** – Specifies if the direct super classes should be retrieved (True) or if the all super classes (ancestors) classes should be retrieved (False).

#### Returns

If **direct** is True, each class C where the set of reasoner axioms entails `DirectSubClassOf(ce, C)`. If **direct** is False, each class C where set of reasoner axioms entails `StrictSubClassOf(ce, C)`. If **ce** is equivalent to `owl:Thing` then nothing will be returned.

**data\_property\_ranges** (*pe: owlapy.owl\_property.OWLDataProperty, direct: bool = False*)  
 → `Iterable[owlapy.owl_data_ranges.OWLDataRange]`

Gets the data ranges that are the direct or indirect ranges of this property with respect to the imports closure of the root ontology.

#### Parameters

- **pe** – The property expression whose ranges are to be retrieved.
- **direct** – Specifies if the direct ranges should be retrieved (True), or if all ranges should be retrieved (False).

Returns:

**all\_data\_property\_values** (*pe: owlapy.owl\_property.OWLDataProperty, direct: bool = True*)  
 → `Iterable[owlapy.owl_literal.OWLLiteral]`

Gets all values for the given data property expression that appear in the knowledge base.

#### Parameters

- **pe** – The data property expression whose values are to be retrieved
- **direct** – Specifies if only the direct values of the data property **pe** should be retrieved (True), or if the values of sub properties of **pe** should be taken into account (False).

#### Returns

A set of OWLLiterals containing literals such that for each literal **l** in the set, the set of reasoner axioms entails `DataPropertyAssertion(pe ind l)` for any **ind**.

**ind\_data\_properties** (*ind: owlapy.owl\_individual.OWLNamedIndividual, direct: bool = True*)  
 → `Iterable[owlapy.owl_property.OWLDataProperty]`

Gets all data properties for the given individual that appear in the knowledge base.

#### Parameters

- **ind** – The named individual whose data properties are to be retrieved
- **direct** – Specifies if the direct data properties should be retrieved (True), or if all data properties should be retrieved (False), so that sub properties are taken into account.

#### Returns

All data properties **pe** where the set of reasoner axioms entails `DataPropertyAssertion(pe ind l)` for atleast one **l**.

**ind\_object\_properties** (*ind: owlapy.owl\_individual.OWLNamedIndividual, direct: bool = True*)  
 → `Iterable[owlapy.owl_property.OWLObjectProperty]`

Gets all object properties for the given individual that appear in the knowledge base.

#### Parameters

- **ind** – The named individual whose object properties are to be retrieved
- **direct** – Specifies if the direct object properties should be retrieved (True), or if all object properties should be retrieved (False), so that sub properties are taken into account.

### Returns

All data properties *pe* where the set of reasoner axioms entails `ObjectPropertyAssertion(pe ind ind2)` for atleast one *ind2*.

## owlapy.class\_expression

OWL Class Expressions [https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions) `ClassExpression` :=

`owl_class.py:`      `Class`    `nary_boolean_expression.py:`      `ObjectIntersectionOf`,    `ObjectUnionOf`  
`class_expression.py:` `ObjectComplementOf`  
  
`restriction.py:`      `ObjectOneOf`,    `ObjectSomeValuesFrom`,    `ObjectAllValuesFrom`,    `ObjectHas-`  
                         `Value`, `ObjectHasSelf`,    `ObjectMinCardinality`,    `ObjectMaxCardinality`,    `ObjectExactCardinality`,    `Data-`  
                         `SomeValuesFrom`,    `DataAllValuesFrom`,    `DataHasValue`,    `DataMinCardinality`,    `DataMaxCardinality`,  
                         `DataExactCardinality`

## Submodules

### owlapy.class\_expression.class\_expression

OWL Base Classes Expressions

## Classes

<i>OWLClassExpression</i>	OWL Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties;
<i>OWLANonymousClassExpression</i>	A Class Expression which is not a named Class.
<i>OWLBooleanClassExpression</i>	Represent an anonymous boolean class expression.
<i>OWLObjectComplementOf</i>	Represents an <code>ObjectComplementOf</code> class expression in the OWL 2 Specification.

## Module Contents

**class** `owlapy.class_expression.class_expression.OWLClassExpression`

Bases: `owlapy.owl_data_ranges.OWLPropertyRange`

OWL Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties; individuals satisfying these conditions are said to be instances of the respective class expressions. In the structural specification of OWL 2, class expressions are represented by `ClassExpression`. ([https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions))

**\_\_slots\_\_** = ()

**abstract** `is_owl_thing()` → bool

Determines if this expression is the built in class `owl:Thing`. This method does not determine if the class is equivalent to `owl:Thing`.

### Returns

`Thing`.

### Return type

True if this expression is owl

**abstract is\_owl\_nothing()** → bool

Determines if this expression is the built in class owl:Nothing. This method does not determine if the class is equivalent to owl:Nothing.

**abstract get\_object\_complement\_of()** → *OWLObjectComplementOf*

Gets the object complement of this class expression.

**Returns**

A class expression that is the complement of this class expression.

**abstract get\_nnf()** → *OWLClassExpression*

Gets the negation normal form of the complement of this expression.

**Returns**

A expression that represents the NNF of the complement of this expression.

**class** owlapy.class\_expression.class\_expression.**OWLAnonymousClassExpression**

Bases: *OWLClassExpression*

A Class Expression which is not a named Class.

**is\_owl\_nothing()** → bool

Determines if this expression is the built in class owl:Nothing. This method does not determine if the class is equivalent to owl:Nothing.

**is\_owl\_thing()** → bool

Determines if this expression is the built in class owl:Thing. This method does not determine if the class is equivalent to owl:Thing.

**Returns**

Thing.

**Return type**

True if this expression is owl

**get\_object\_complement\_of()** → *OWLObjectComplementOf*

Gets the object complement of this class expression.

**Returns**

A class expression that is the complement of this class expression.

**get\_nnf()** → *OWLClassExpression*

Gets the negation normal form of the complement of this expression.

**Returns**

A expression that represents the NNF of the complement of this expression.

**class** owlapy.class\_expression.class\_expression.**OWLBooleanClassExpression**

Bases: *OWLAnonymousClassExpression*

Represent an anonymous boolean class expression.

**\_\_slots\_\_** = ()

**class** owlapy.class\_expression.class\_expression.**OWLObjectComplementOf**(  
    *op: OWLClassExpression*)

Bases: *OWLBooleanClassExpression*, *owlapy.meta\_classes.HasOperands[OWLClassExpression]*

Represents an ObjectComplementOf class expression in the OWL 2 Specification.

```

__slots__ = '_operand'

type_index: Final = 3003

get_operand() → OWLClassExpression

```

**Returns**  
The wrapped expression.

```

operands() → Iterable[OWLClassExpression]

```

Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

**Returns**  
The operands.

```

__repr__()

__eq__(other)

__hash__()

```

## owlapy.class\_expression.nary\_boolean\_expression

OWL nary boolean expressions

### Classes

<i>OWLNaryBooleanClassExpression</i>	OWLNaryBooleanClassExpression.
<i>OWLObjectUnionOf</i>	A union class expression <code>ObjectUnionOf( CE1 ... CEn )</code> contains all individuals that are instances
<i>OWLObjectIntersectionOf</i>	An intersection class expression <code>ObjectIntersectionOf( CE1 ... CEn )</code> contains all individuals that are instances

### Module Contents

```

class owlapy.class_expression.nary_boolean_expression.
    OWLNaryBooleanClassExpression (
        operands: Iterable[owlapy.class_expression.class_expression.OWLClassExpression])
    Bases:
        owlapy.class_expression.class_expression.OWLBooleanClassExpression,
        owlapy.meta_classes.HasOperands[owlapy.class_expression.class_expression.
            OWLClassExpression]
    OWLNaryBooleanClassExpression.
    __slots__ = ()
    operands() → Iterable[owlapy.class_expression.class_expression.OWLClassExpression]
        Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.
    Returns
        The operands.
    __repr__()
    __eq__(other)

```

`__hash__()`

```
class owlapy.class_expression.nary_boolean_expression.OWLObjectUnionOf(
    operands: Iterable[owlapy.class_expression.class_expression.OWLClassExpression])
```

Bases: *OWLNaryBooleanClassExpression*

A union class expression `ObjectUnionOf( CE1 ... CEn )` contains all individuals that are instances of at least one class expression `CEi` for  $1 \leq i \leq n$ . ([https://www.w3.org/TR/owl2-syntax/#Union\\_of\\_Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Union_of_Class_Expressions))

`__slots__ = '_operands'`

`type_index: Final = 3002`

```
class owlapy.class_expression.nary_boolean_expression.OWLObjectIntersectionOf(
    operands: Iterable[owlapy.class_expression.class_expression.OWLClassExpression])
```

Bases: *OWLNaryBooleanClassExpression*

An intersection class expression `ObjectIntersectionOf( CE1 ... CEn )` contains all individuals that are instances of all class expressions `CEi` for  $1 \leq i \leq n$ . ([https://www.w3.org/TR/owl2-syntax/#Intersection\\_of\\_Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Intersection_of_Class_Expressions))

`__slots__ = '_operands'`

`type_index: Final = 3001`

## owlapy.class\_expression.owl\_class

OWL Class

### Classes

<i>OWLClass</i>	An OWL 2 named Class. Classes can be understood as sets of individuals.
-----------------	---

## Module Contents

```
class owlapy.class_expression.owl_class.OWLClass(iri: owlapy.iri.IRI | str)
```

Bases: *owlapy.class\_expression.class\_expression.OWLClassExpression*, *owlapy.owl\_object.OWLEntity*

An OWL 2 named Class. Classes can be understood as sets of individuals. (<https://www.w3.org/TR/owl2-syntax/#Classes>)

`__slots__ = ('_iri', '_is_nothing', '_is_thing')`

`type_index: Final = 1001`

property `iri`: *owlapy.iri.IRI*

Gets the IRI of this object.

#### Returns

The IRI of this object.

property `str`

Gets the string representation of this object

**Returns**

The IRI as string

**property reminder:** `str`

The reminder of the IRI

**is\_owl\_thing()** → bool

Determines if this expression is the built in class owl:Thing. This method does not determine if the class is equivalent to owl:Thing.

**Returns**

Thing.

**Return type**

True if this expression is owl

**is\_owl\_nothing()** → bool

Determines if this expression is the built in class owl:Nothing. This method does not determine if the class is equivalent to owl:Nothing.

**get\_object\_complement\_of()** → *owlapy.class\_expression.class\_expression.OWLObjectComplementOf*

Gets the object complement of this class expression.

**Returns**

A class expression that is the complement of this class expression.

**get\_nnf()** → *OWLClass*

Gets the negation normal form of the complement of this expression.

**Returns**

A expression that represents the NNF of the complement of this expression.

**owlapy.class\_expression.restriction**

OWL Restrictions

**Attributes**

*Literals*





## Classes

<i>OWLRestriction</i>	Represents an Object Property Restriction or Data Property Restriction in the OWL 2 specification.
<i>OWLHasValueRestriction</i>	Represent a HasValue restriction in the OWL 2
<i>OWLObjectRestriction</i>	Represents an Object Property Restriction in the OWL 2 specification.
<i>OWLQuantifiedRestriction</i>	Represents a quantified restriction.
<i>OWLCardinalityRestriction</i>	Base interface for owl min and max cardinality restriction.
<i>OWLQuantifiedObjectRestriction</i>	Represents a quantified object restriction.
<i>OWLObjectCardinalityRestriction</i>	Represents Object Property Cardinality Restrictions in the OWL 2 specification.
<i>OWLObjectMinCardinality</i>	A minimum cardinality expression <i>ObjectMinCardinality</i> ( <i>n</i> OPE CE ) consists of a nonnegative integer <i>n</i> , an object
<i>OWLObjectMaxCardinality</i>	A maximum cardinality expression <i>ObjectMaxCardinality</i> ( <i>n</i> OPE CE ) consists of a nonnegative integer <i>n</i> , an object
<i>OWLObjectExactCardinality</i>	An exact cardinality expression <i>ObjectExactCardinality</i> ( <i>n</i> OPE CE ) consists of a nonnegative integer <i>n</i> , an object
<i>OWLObjectSomeValuesFrom</i>	An existential class expression <i>ObjectSomeValuesFrom</i> ( OPE CE ) consists of an object property expression OPE and
<i>OWLObjectAllValuesFrom</i>	A universal class expression <i>ObjectAllValuesFrom</i> ( OPE CE ) consists of an object property expression OPE and a
<i>OWLObjectHasSelf</i>	A self-restriction <i>ObjectHasSelf</i> ( OPE ) consists of an object property expression OPE,
<i>OWLObjectHasValue</i>	A has-value class expression <i>ObjectHasValue</i> ( OPE <i>a</i> ) consists of an object property expression OPE and an
<i>OWLObjectOneOf</i>	An enumeration of individuals <i>ObjectOneOf</i> ( <i>a</i> <sub>1</sub> ... <i>a</i> <sub><i>n</i></sub> ) contains exactly the individuals <i>a</i> <sub><i>i</i></sub> with 1 ≤ <i>i</i> ≤ <i>n</i> .
<i>OWLDataRestriction</i>	Represents a Data Property Restriction.
<i>OWLQuantifiedDataRestriction</i>	Represents a quantified data restriction.
<i>OWLDataCardinalityRestriction</i>	Represents Data Property Cardinality Restrictions.
<i>OWLDataMinCardinality</i>	A minimum cardinality expression <i>DataMinCardinality</i> ( <i>n</i> DPE DR ) consists of a nonnegative integer <i>n</i> , a data
<i>OWLDataMaxCardinality</i>	A maximum cardinality expression <i>ObjectMaxCardinality</i> ( <i>n</i> OPE CE ) consists of a nonnegative integer <i>n</i> , an object
<i>OWLDataExactCardinality</i>	An exact cardinality expression <i>ObjectExactCardinality</i> ( <i>n</i> OPE CE ) consists of a nonnegative integer <i>n</i> , an
<i>OWLDataSomeValuesFrom</i>	An existential class expression <i>DataSomeValuesFrom</i> ( DPE <sub>1</sub> ... DPE <sub><i>n</i></sub> DR ) consists of <i>n</i> data property expressions
<i>OWLDataAllValuesFrom</i>	A universal class expression <i>DataAllValuesFrom</i> ( DPE <sub>1</sub> ... DPE <sub><i>n</i></sub> DR ) consists of <i>n</i> data property expressions DPE <sub><i>i</i></sub> ,
<i>OWLDataHasValue</i>	A has-value class expression <i>DataHasValue</i> ( DPE <i>l</i> <sub><i>t</i></sub> ) consists of a data property expression DPE and a literal <i>l</i> <sub><i>t</i></sub> ,
<i>OWLDataOneOf</i>	An enumeration of literals <i>DataOneOf</i> ( <i>l</i> <sub><i>t</i></sub> <sub>1</sub> ... <i>l</i> <sub><i>t</i></sub> <sub><i>n</i></sub> ) contains exactly the explicitly specified literals <i>l</i> <sub><i>t</i></sub> <sub><i>i</i></sub> with
<i>OWLDatatypeRestriction</i>	A datatype restriction <i>DatatypeRestriction</i> ( DT F <sub>1</sub> <i>l</i> <sub><i>t</i></sub> <sub>1</sub> ... F <sub><i>n</i></sub> <i>l</i> <sub><i>t</i></sub> <sub><i>n</i></sub> ) consists of a unary datatype DT and <i>n</i> pairs
<i>OWLFacetRestriction</i>	A facet restriction is used to restrict a particular datatype.

## Module Contents

`owlapy.class_expression.restriction.Literals`

**class** `owlapy.class_expression.restriction.OWLRestriction`

Bases: `owlapy.class_expression.class_expression.OWLAnonymousClassExpression`

Represents an Object Property Restriction or Data Property Restriction in the OWL 2 specification.

`__slots__ = ()`

**abstract** `get_property()` → `owlapy.owl_property.OWLPropertyExpression`

**Returns**

Property being restricted.

`is_data_restriction()` → bool

Determines if this is a data restriction.

**Returns**

True if this is a data restriction.

`is_object_restriction()` → bool

Determines if this is an object restriction.

**Returns**

True if this is an object restriction.

**class** `owlapy.class_expression.restriction.OWLHasValueRestriction(value: _T)`

Bases: `Generic[_T]`, `OWLRestriction`, `owlapy.meta_classes.HasFiller[_T]`

Represent a HasValue restriction in the OWL 2

**Parameters**

**value** – The value type `_T`.

`__slots__ = ()`

`__eq__(other)`

`__hash__()`

`get_filler()` → `_T`

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**

the value

**class** `owlapy.class_expression.restriction.OWLObjectRestriction`

Bases: `OWLRestriction`

Represents an Object Property Restriction in the OWL 2 specification.

`__slots__ = ()`

`is_object_restriction()` → bool

Determines if this is an object restriction.

**Returns**

True if this is an object restriction.

**abstract** `get_property()`  $\rightarrow$  *owlapy.owl\_property.OWLObjectPropertyExpression*

**Returns**

Property being restricted.

**class** `owlapy.class_expression.restriction.OWLQuantifiedRestriction`

Bases: `Generic[_T]`, *OWLRestriction*, *owlapy.meta\_classes.HasFiller[\_T]*

Represents a quantified restriction.

**Parameters**

`_T` – value type

`__slots__` = ()

**class** `owlapy.class_expression.restriction.OWLCardinalityRestriction` (*cardinality: int*,  
*filler: \_F*)

Bases: `Generic[_F]`, *OWLQuantifiedRestriction[\_F]*, *owlapy.meta\_classes.HasCardinality*

Base interface for owl min and max cardinality restriction.

**Parameters**

`_F` – Type of filler.

`__slots__` = ()

**get\_cardinality()**  $\rightarrow$  int

Gets the cardinality of a restriction.

**Returns**

The cardinality. A non-negative integer.

**get\_filler()**  $\rightarrow$  `_F`

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**

the value

**class** `owlapy.class_expression.restriction.OWLQuantifiedObjectRestriction` (  
*filler: owlapy.class\_expression.class\_expression.OWLClassExpression*)

Bases: *OWLQuantifiedRestriction[owlapy.class\_expression.class\_expression.OWLClassExpression]*, *OWLObjectRestriction*

Represents a quantified object restriction.

`__slots__` = ()

**get\_filler()**  $\rightarrow$  *owlapy.class\_expression.class\_expression.OWLClassExpression*

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**

the value

**class** `owlapy.class_expression.restriction.OWLObjectCardinalityRestriction` (  
*cardinality: int*, *property: owlapy.owl\_property.OWLObjectPropertyExpression*,  
*filler: owlapy.class\_expression.class\_expression.OWLClassExpression*)

Bases: `OWLCardinalityRestriction[owlapy.class_expression.class_expression.OWLClassExpression]`, `OWLQuantifiedObjectRestriction`

Represents Object Property Cardinality Restrictions in the OWL 2 specification.

`__slots__ = ()`

`get_property()` → `owlapy.owl_property.OWLObjectPropertyExpression`

**Returns**

Property being restricted.

`__repr__()`

`__eq__(other)`

`__hash__()`

```
class owlapy.class_expression.restriction.OWLObjectMinCardinality (cardinality: int,
    property: owlapy.owl_property.OWLObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OWLClassExpression)
```

Bases: `OWLObjectCardinalityRestriction`

A minimum cardinality expression `ObjectMinCardinality( n OPE CE )` consists of a nonnegative integer `n`, an object property expression `OPE`, and a class expression `CE`, and it contains all those individuals that are connected by `OPE` to at least `n` different individuals that are instances of `CE`. ([https://www.w3.org/TR/owl2-syntax/#Minimum\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Minimum_Cardinality))

`__slots__ = ('_cardinality', '_filler', '_property')`

`type_index: Final = 3008`

```
class owlapy.class_expression.restriction.OWLObjectMaxCardinality (cardinality: int,
    property: owlapy.owl_property.OWLObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OWLClassExpression)
```

Bases: `OWLObjectCardinalityRestriction`

A maximum cardinality expression `ObjectMaxCardinality( n OPE CE )` consists of a nonnegative integer `n`, an object property expression `OPE`, and a class expression `CE`, and it contains all those individuals that are connected by `OPE`

to at most `n` different individuals that are instances of `CE`. ([https://www.w3.org/TR/owl2-syntax/#Maximum\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Maximum_Cardinality))

`__slots__ = ('_cardinality', '_filler', '_property')`

`type_index: Final = 3010`

```
class owlapy.class_expression.restriction.OWLObjectExactCardinality (cardinality: int,
    property: owlapy.owl_property.OWLObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OWLClassExpression)
```

Bases: `OWLObjectCardinalityRestriction`

**An exact cardinality expression `ObjectExactCardinality( n OPE CE )` consists of a nonnegative integer `n`, an object**

**property expression `OPE`, and a class expression `CE`, and it contains all those individuals that are connected by to exactly `n` different individuals that are instances of `CE`.**

([https://www.w3.org/TR/owl2-syntax/#Exact\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Exact_Cardinality))

```
__slots__ = ('_cardinality', '_filler', '_property')
```

```
type_index: Final = 3009
```

```
as_intersection_of_min_max()
```

→ *owlapy.class\_expression.nary\_boolean\_expression.OwlObjectIntersectionOf*

Obtains an equivalent form that is a conjunction of a min cardinality and max cardinality restriction.

#### Returns

The semantically equivalent but structurally simpler form ( $= 1 \text{ R } C$ )  $\Rightarrow 1 \text{ R } C$  and  $\leq 1 \text{ R } C$ .

```
class owlapy.class_expression.restriction.OwlObjectSomeValuesFrom(  
    property: owlapy.owl_property.OwlObjectPropertyExpression,  
    filler: owlapy.class_expression.class_expression.OwlClassExpression)
```

Bases: *OwlQuantifiedObjectRestriction*

An existential class expression *ObjectSomeValuesFrom*( OPE CE ) consists of an object property expression OPE and a class expression CE, and it contains all those individuals that are connected by OPE to an individual that is an instance of CE.

```
__slots__ = ('_property', '_filler')
```

```
type_index: Final = 3005
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
get_property() → owlapy.owl_property.OwlObjectPropertyExpression
```

#### Returns

Property being restricted.

```
class owlapy.class_expression.restriction.OwlObjectAllValuesFrom(  
    property: owlapy.owl_property.OwlObjectPropertyExpression,  
    filler: owlapy.class_expression.class_expression.OwlClassExpression)
```

Bases: *OwlQuantifiedObjectRestriction*

A universal class expression *ObjectAllValuesFrom*( OPE CE ) consists of an object property expression OPE and a class expression CE, and it contains all those individuals that are connected by OPE only to individuals that are instances of CE. ([https://www.w3.org/TR/owl2-syntax/#Universal\\_Quantification](https://www.w3.org/TR/owl2-syntax/#Universal_Quantification))

```
__slots__ = ('_property', '_filler')
```

```
type_index: Final = 3006
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
get_property() → owlapy.owl_property.OwlObjectPropertyExpression
```

#### Returns

Property being restricted.

```

class owlapy.class_expression.restriction.OwlObjectHasSelf (
    property: owlapy.owl_property.OwlObjectPropertyExpression)
    Bases: OwlObjectRestriction

    A self-restriction ObjectHasSelf( OPE ) consists of an object property expression OPE, and it contains all those
    individuals that are connected by OPE to themselves. (https://www.w3.org/TR/owl2-syntax/#Self-Restriction)

    __slots__ = '_property'

    type_index: Final = 3011

    get_property() → owlapy.owl_property.OwlObjectPropertyExpression

        Returns
            Property being restricted.

    __eq__(other)

    __hash__()

    __repr__()

class owlapy.class_expression.restriction.OwlObjectHasValue (
    property: owlapy.owl_property.OwlObjectPropertyExpression,
    individual: owlapy.owl_individual.OwlIndividual)
    Bases: OwlHasValueRestriction[owlapy.owl_individual.OwlIndividual], OwlObjectRestriction

    A has-value class expression ObjectHasValue( OPE a ) consists of an object property expression OPE and an
    individual a, and it contains all those individuals that are connected by OPE to a. Each such class expression
    can be seen as a syntactic shortcut for the class expression ObjectSomeValuesFrom( OPE ObjectOneOf( a ) ).
    (https://www.w3.org/TR/owl2-syntax/#Individual\_Value\_Restriction)

    __slots__ = ('_property', '_v')

    type_index: Final = 3007

    get_property() → owlapy.owl_property.OwlObjectPropertyExpression

        Returns
            Property being restricted.

    as_some_values_from() → owlapy.class_expression.class_expression.OwlClassExpression

        A convenience method that obtains this restriction as an existential restriction with a nominal filler.

        Returns
            The existential equivalent of this value restriction.  $\text{simp}(\text{HasValue}(p\ a)) = \text{some}(p\ \{a\})$ .

    __repr__()

class owlapy.class_expression.restriction.OwlObjectOneOf (
    values: owlapy.owl_individual.OwlIndividual | Iterable[owlapy.owl_individual.OwlIndividual])
    Bases: owlapy.class_expression.class_expression.OwlAnonymousClassExpression, owlapy.
    meta_classes.HasOperands[owlapy.owl_individual.OwlIndividual]

    An enumeration of individuals ObjectOneOf( a1 ... an ) contains exactly the individuals ai with  $1 \leq i \leq n$ . (https://www.w3.org/TR/owl2-syntax/#Enumeration\_of\_Individuals)

    __slots__ = '_values'

```

**type\_index:** `Final = 3004`

**individuals()** → `Iterable[owlapy.owl_individual.OWLIndividual]`  
 Gets the individuals that are in the oneOf. These individuals represent the exact instances (extension) of this class expression.

**Returns**  
 The individuals that are the values of this `{@code ObjectOneOf}` class expression.

**operands()** → `Iterable[owlapy.owl_individual.OWLIndividual]`  
 Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

**Returns**  
 The operands.

**as\_object\_union\_of()** → `owlapy.class_expression.class_expression.OWLClassExpression`  
 Simplifies this enumeration to a union of singleton nominals.

**Returns**  
 This enumeration in a more standard DL form.  $\text{simp}(\{a\}) = \{a\}$   $\text{simp}(\{a_0, \dots, \{a_n\}) = \text{unionOf}(\{a_0\}, \dots, \{a_n\})$

**\_\_hash\_\_()**

**\_\_eq\_\_(other)**

**\_\_repr\_\_()**

**class owlapy.class\_expression.restriction.OWLDataRestriction**  
 Bases: `OWLRestriction`  
 Represents a Data Property Restriction.

**\_\_slots\_\_ = ()**

**is\_data\_restriction()** → `bool`  
 Determines if this is a data restriction.

**Returns**  
 True if this is a data restriction.

**class owlapy.class\_expression.restriction.OWLQuantifiedDataRestriction(**  
   `filler: owlapy.owl_data_ranges.OWLDataRange)`  
 Bases: `OWLQuantifiedRestriction[owlapy.owl_data_ranges.OWLDataRange], OWLDataRestriction`  
 Represents a quantified data restriction.

**\_\_slots\_\_ = ()**

**get\_filler()** → `owlapy.owl_data_ranges.OWLDataRange`  
 Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**  
 the value



```

class owlapy.class_expression.restriction.OWLDataCardinalityRestriction(
    cardinality: int, property: owlapy.owl_property.OWLDataPropertyExpression,
    filler: owlapy.owl_data_ranges.OWLDataRange)

Bases: OWLCardinalityRestriction[owlapy.owl_data_ranges.OWLDataRange], OWLQuantified-
DataRestriction, OWLDataRestriction

Represents Data Property Cardinality Restrictions.

__slots__ = ()

__hash__()

__repr__()

__eq__(other)

get_property() → owlapy.owl_property.OWLDataPropertyExpression

Returns
    Property being restricted.

class owlapy.class_expression.restriction.OWLDataMinCardinality(cardinality: int,
    property: owlapy.owl_property.OWLDataPropertyExpression,
    filler: owlapy.owl_data_ranges.OWLDataRange)

Bases: OWLDataCardinalityRestriction

A minimum cardinality expression DataMinCardinality( n DPE DR ) consists of a nonnegative integer n, a data
property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by
DPE to at least n different literals in DR. (https://www.w3.org/TR/owl2-syntax/#Minimum\_Cardinality)

__slots__ = ('_cardinality', '_filler', '_property')

type_index: Final = 3015

class owlapy.class_expression.restriction.OWLDataMaxCardinality(cardinality: int,
    property: owlapy.owl_property.OWLDataPropertyExpression,
    filler: owlapy.owl_data_ranges.OWLDataRange)

Bases: OWLDataCardinalityRestriction

A maximum cardinality expression ObjectMaxCardinality( n OPE CE ) consists of a nonnegative integer n, an
object property expression OPE, and a class expression CE, and it contains all those individuals that are connected
by OPE to at most n different individuals that are instances of CE. (https://www.w3.org/TR/owl2-syntax/#Maximum\_Cardinality)

__slots__ = ('_cardinality', '_filler', '_property')

type_index: Final = 3017

class owlapy.class_expression.restriction.OWLDataExactCardinality(cardinality: int,
    property: owlapy.owl_property.OWLDataPropertyExpression,
    filler: owlapy.owl_data_ranges.OWLDataRange)

Bases: OWLDataCardinalityRestriction

An exact cardinality expression ObjectExactCardinality( n OPE CE ) consists of a nonnegative integer n, an object
property expression OPE, and a class expression CE, and it contains all those individuals that are connected
by OPE to exactly n different individuals that are instances of CE (https://www.w3.org/TR/owl2-syntax/#Exact\_Cardinality)

__slots__ = ('_cardinality', '_filler', '_property')

```

**type\_index:** Final = 3016

**as\_intersection\_of\_min\_max()**

→ *owlapy.class\_expression.nary\_boolean\_expression.OwlObjectIntersectionOf*

Obtains an equivalent form that is a conjunction of a min cardinality and max cardinality restriction.

#### Returns

The semantically equivalent but structurally simpler form  $(= 1 \text{ R D}) = \geq 1 \text{ R D}$  and  $\leq 1 \text{ R D}$ .

```
class owlapy.class_expression.restriction.OwlDataSomeValuesFrom(  
    property: owlapy.owl_property.OwlDataPropertyExpression,  
    filler: owlapy.owl_data_ranges.OwlDataRange)
```

Bases: *OwlQuantifiedDataRestriction*

An existential class expression `DataSomeValuesFrom( DPE1 ... DPEn DR )` consists of  $n$  data property expressions  $DPE_i$ ,  $1 \leq i \leq n$ , and a data range  $DR$  whose arity must be  $n$ . Such a class expression contains all those individuals that are connected by  $DPE_i$  to literals  $l_{ti}$ ,  $1 \leq i \leq n$ , such that the tuple  $( l_{t1} , \dots , l_{tn} )$  is in  $DR$ . A class expression of the form `DataSomeValuesFrom( DPE DR )` can be seen as a syntactic shortcut for the class expression `DataMinCardinality( 1 DPE DR )`. ([https://www.w3.org/TR/owl2-syntax/#Existential\\_Quantification\\_2](https://www.w3.org/TR/owl2-syntax/#Existential_Quantification_2))

**\_\_slots\_\_** = **'\_property'**

**type\_index:** Final = 3012

**\_\_repr\_\_()**

**\_\_eq\_\_**(*other*)

**\_\_hash\_\_()**

**get\_property()** → *owlapy.owl\_property.OwlDataPropertyExpression*

#### Returns

Property being restricted.

```
class owlapy.class_expression.restriction.OwlDataAllValuesFrom(  
    property: owlapy.owl_property.OwlDataPropertyExpression,  
    filler: owlapy.owl_data_ranges.OwlDataRange)
```

Bases: *OwlQuantifiedDataRestriction*

A universal class expression `DataAllValuesFrom( DPE1 ... DPEn DR )` consists of  $n$  data property expressions  $DPE_i$ ,  $1 \leq i \leq n$ , and a data range  $DR$  whose arity must be  $n$ . Such a class expression contains all those individuals that

**are connected by  $DPE_i$  only to literals  $l_{ti}$ ,  $1 \leq i \leq n$ , such that each tuple  $( l_{t1} , \dots , l_{tn} )$  is in  $DR$ .**

#### A class

expression of the form `DataAllValuesFrom( DPE DR )` can be seen as a syntactic shortcut for the class expression `DataMaxCardinality( 0 DPE DataComplementOf( DR ) )`. ([https://www.w3.org/TR/owl2-syntax/#Universal\\_Quantification\\_2](https://www.w3.org/TR/owl2-syntax/#Universal_Quantification_2))

**\_\_slots\_\_** = **'\_property'**

**type\_index:** Final = 3013

**\_\_repr\_\_()**

**\_\_eq\_\_**(*other*)

**\_\_hash\_\_()**

**get\_property()** → *owlapy.owl\_property.OWLDataPropertyExpression*

**Returns**

Property being restricted.

```
class owlapy.class_expression.restriction.OWLDataHasValue(  
    property: owlapy.owl_property.OWLDataPropertyExpression,  
    value: owlapy.owl_literal.OWLLiteral)
```

Bases: *OWLHasValueRestriction[owlapy.owl\_literal.OWLLiteral], OWLDataRestriction*

A has-value class expression `DataHasValue( DPE lt )` consists of a data property expression DPE and a literal lt, and it contains all those individuals that are connected by DPE to lt. Each such class expression can be seen as a syntactic shortcut for the class expression `DataSomeValuesFrom( DPE DataOneOf( lt ) )`. ([https://www.w3.org/TR/owl2-syntax/#Literal\\_Value\\_Restriction](https://www.w3.org/TR/owl2-syntax/#Literal_Value_Restriction))

```
__slots__ = '_property'
```

```
type_index: Final = 3014
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

**as\_some\_values\_from()** → *owlapy.class\_expression.class\_expression.OWLClassExpression*

A convenience method that obtains this restriction as an existential restriction with a nominal filler.

**Returns**

The existential equivalent of this value restriction.  $\text{simp}(\text{HasValue}(p\ a)) = \text{some}(p\ \{a\})$ .

**get\_property()** → *owlapy.owl\_property.OWLDataPropertyExpression*

**Returns**

Property being restricted.

```
class owlapy.class_expression.restriction.OWLDataOneOf(  
    values: owlapy.owl_literal.OWLLiteral | Iterable[owlapy.owl_literal.OWLLiteral])
```

Bases: *owlapy.owl\_data\_ranges.OWLDataRange, owlapy.meta\_classes.HasOperands[owlapy.owl\_literal.OWLLiteral]*

An enumeration of literals `DataOneOf( lt1 ... ltn )` contains exactly the explicitly specified literals `lti` with  $1 \leq i \leq n$ . The resulting data range has arity one. ([https://www.w3.org/TR/owl2-syntax/#Enumeration\\_of\\_Literals](https://www.w3.org/TR/owl2-syntax/#Enumeration_of_Literals))

```
type_index: Final = 4003
```

```
__repr__()
```

```
__hash__()
```

```
__eq__(other)
```

**values()** → *Iterable[owlapy.owl\_literal.OWLLiteral]*

Gets the values that are in the oneOf.

**Returns**

The values of this `{@code DataOneOf}` class expression.

**operands** () → Iterable[owlapy.owl\_literal.OWLLiteral]

Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

#### Returns

The operands.

```
class owlapy.class_expression.restriction.OWLDatatypeRestriction(  
    type_: owlapy.owl_datatype.OWLDatatype,  
    facet_restrictions: OWLFacetRestriction | Iterable[OWLFacetRestriction])
```

Bases: owlapy.owl\_data\_ranges.OWLDataRange

A datatype restriction DatatypeRestriction( DT F1 lti1 ... Fn ltn ) consists of a unary datatype DT and n pairs ( Fi , lti ). The resulting data range is unary and is obtained by restricting the value space of DT according to the semantics of all ( Fi , vi ) (multiple pairs are interpreted conjunctively), where vi are the data values of the literals lti. ([https://www.w3.org/TR/owl2-syntax/#Datatype\\_Restrictions](https://www.w3.org/TR/owl2-syntax/#Datatype_Restrictions))

```
__slots__ = ('_type', '_facet_restrictions')
```

```
type_index: Final = 4006
```

```
get_datatype() → owlapy.owl_datatype.OWLDatatype
```

```
get_facet_restrictions() → Sequence[OWLFacetRestriction]
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.class_expression.restriction.OWLFacetRestriction(  
    facet: owlapy.vocab.OWLFacet, literal: Literals)
```

Bases: owlapy.owl\_object.OWLObject

A facet restriction is used to restrict a particular datatype.

```
__slots__ = ('_facet', '_literal')
```

```
type_index: Final = 4007
```

```
get_facet() → owlapy.vocab.OWLFacet
```

```
get_facet_value() → owlapy.owl_literal.OWLLiteral
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

## Classes

<i>OWLClassExpression</i>	OWL Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties;
<i>OWLAnonymousClassExpression</i>	A Class Expression which is not a named Class.
<i>OWLBooleanClassExpression</i>	Represent an anonymous boolean class expression.

continues on next page

Table 1 – continued from previous page

<i>OWLObjectComplementOf</i>	Represents an ObjectComplementOf class expression in the OWL 2 Specification.
<i>OWLClass</i>	An OWL 2 named Class. Classes can be understood as sets of individuals.
<i>OWLNaryBooleanClassExpression</i>	OWLNaryBooleanClassExpression.
<i>OWLObjectUnionOf</i>	A union class expression ObjectUnionOf( CE1 ... CEn ) contains all individuals that are instances
<i>OWLObjectIntersectionOf</i>	An intersection class expression ObjectIntersectionOf( CE1 ... CEn ) contains all individuals that are instances
<i>OWLRestriction</i>	Represents an Object Property Restriction or Data Property Restriction in the OWL 2 specification.
<i>OWLQuantifiedRestriction</i>	Represents a quantified restriction.
<i>OWLQuantifiedObjectRestriction</i>	Represents a quantified object restriction.
<i>OWLObjectRestriction</i>	Represents an Object Property Restriction in the OWL 2 specification.
<i>OWLHasValueRestriction</i>	Represent a HasValue restriction in the OWL 2
<i>OWLDDataRestriction</i>	Represents a Data Property Restriction.
<i>OWLCardinalityRestriction</i>	Base interface for owl min and max cardinality restriction.
<i>OWLObjectCardinalityRestriction</i>	Represents Object Property Cardinality Restrictions in the OWL 2 specification.
<i>OWLObjectHasSelf</i>	A self-restriction ObjectHasSelf( OPE ) consists of an object property expression OPE,
<i>OWLDDataOneOf</i>	An enumeration of literals DataOneOf( lt1 ... ltn ) contains exactly the explicitly specified literals lti with
<i>OWLQuantifiedDataRestriction</i>	Represents a quantified data restriction.
<i>OWLDDataCardinalityRestriction</i>	Represents Data Property Cardinality Restrictions.
<i>OWLObjectSomeValuesFrom</i>	An existential class expression ObjectSomeValuesFrom( OPE CE ) consists of an object property expression OPE and
<i>OWLObjectAllValuesFrom</i>	A universal class expression ObjectAllValuesFrom( OPE CE ) consists of an object property expression OPE and a
<i>OWLObjectHasValue</i>	A has-value class expression ObjectHasValue( OPE a ) consists of an object property expression OPE and an
<i>OWLDatatypeRestriction</i>	A datatype restriction DatatypeRestriction( DT F1 lt1 ... Fn ltn ) consists of a unary datatype DT and n pairs
<i>OWLFacet</i>	Enumerations for OWL facets.
<i>OWLFacetRestriction</i>	A facet restriction is used to restrict a particular datatype.
<i>OWLObjectMinCardinality</i>	A minimum cardinality expression ObjectMinCardinality( n OPE CE ) consists of a nonnegative integer n, an object
<i>OWLObjectMaxCardinality</i>	A maximum cardinality expression ObjectMaxCardinality( n OPE CE ) consists of a nonnegative integer n, an object
<i>OWLObjectExactCardinality</i>	An exact cardinality expression ObjectExactCardinality( n OPE CE ) consists of a nonnegative integer n, an object
<i>OWLDDataSomeValuesFrom</i>	An existential class expression DataSomeValuesFrom( DPE1 ... DPEn DR ) consists of n data property expressions
<i>OWLDDataAllValuesFrom</i>	A universal class expression DataAllValuesFrom( DPE1 ... DPEn DR ) consists of n data property expressions DPEi,

continues on next page

Table 1 – continued from previous page

<i>OWLDataHasValue</i>	A has-value class expression <i>DataHasValue</i> ( DPE lt ) consists of a data property expression DPE and a literal lt,
<i>OWLDataMinCardinality</i>	A minimum cardinality expression <i>DataMinCardinality</i> ( n DPE DR ) consists of a nonnegative integer n, a data
<i>OWLDataMaxCardinality</i>	A maximum cardinality expression <i>ObjectMaxCardinality</i> ( n OPE CE ) consists of a nonnegative integer n, an object
<i>OWLDataExactCardinality</i>	An exact cardinality expression <i>ObjectExactCardinality</i> ( n OPE CE ) consists of a nonnegative integer n, an
<i>OWLObjectOneOf</i>	An enumeration of individuals <i>ObjectOneOf</i> ( a1 ... an ) contains exactly the individuals ai with $1 \leq i \leq n$ .

## Package Contents

**class** owlapy.class\_expression.OWLClassExpression

Bases: *owlapy.owl\_data\_ranges.OWLPropertyRange*

OWL Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties; individuals satisfying these conditions are said to be instances of the respective class expressions. In the structural specification of OWL 2, class expressions are represented by *ClassExpression*. ([https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions))

**\_\_slots\_\_** = ()

**abstract is\_owl\_thing**() → bool

Determines if this expression is the built in class owl:Thing. This method does not determine if the class is equivalent to owl:Thing.

**Returns**

Thing.

**Return type**

True if this expression is owl

**abstract is\_owl\_nothing**() → bool

Determines if this expression is the built in class owl:Nothing. This method does not determine if the class is equivalent to owl:Nothing.

**abstract get\_object\_complement\_of**() → *OWLObjectComplementOf*

Gets the object complement of this class expression.

**Returns**

A class expression that is the complement of this class expression.

**abstract get\_nnf**() → *OWLClassExpression*

Gets the negation normal form of the complement of this expression.

**Returns**

A expression that represents the NNF of the complement of this expression.

**class** owlapy.class\_expression.OWLAnonymousClassExpression

Bases: *OWLClassExpression*

A Class Expression which is not a named Class.

**is\_owl\_nothing()** → bool

Determines if this expression is the built in class owl:Nothing. This method does not determine if the class is equivalent to owl:Nothing.

**is\_owl\_thing()** → bool

Determines if this expression is the built in class owl:Thing. This method does not determine if the class is equivalent to owl:Thing.

**Returns**

Thing.

**Return type**

True if this expression is owl

**get\_object\_complement\_of()** → *OWLObjectComplementOf*

Gets the object complement of this class expression.

**Returns**

A class expression that is the complement of this class expression.

**get\_nnf()** → *OWLClassExpression*

Gets the negation normal form of the complement of this expression.

**Returns**

A expression that represents the NNF of the complement of this expression.

**class** owlapy.class\_expression.OWLBooleanClassExpression

Bases: *OWLAnonymousClassExpression*

Represent an anonymous boolean class expression.

**\_\_slots\_\_** = ()

**class** owlapy.class\_expression.OWLObjectComplementOf (*op: OWLClassExpression*)

Bases: *OWLBooleanClassExpression*, owlapy.meta\_classes.HasOperands[*OWLClassExpression*]

Represents an ObjectComplementOf class expression in the OWL 2 Specification.

**\_\_slots\_\_** = '\_operand'

**type\_index: Final** = 3003

**get\_operand()** → *OWLClassExpression*

**Returns**

The wrapped expression.

**operands()** → Iterable[*OWLClassExpression*]

Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

**Returns**

The operands.

**\_\_repr\_\_()**

**\_\_eq\_\_**(*other*)

**\_\_hash\_\_()**

```

class owlapy.class_expression.OWLClass(iri: owlapy.iri.IRI | str)
    Bases: owlapy.class_expression.class_expression.OWLClassExpression, owlapy.
            owl_object.OWLEntity

    An OWL 2 named Class. Classes can be understood as sets of individuals. (https://www.w3.org/TR/owl2-syntax/#Classes)

    __slots__ = ('_iri', '_is_nothing', '_is_thing')

    type_index: Final = 1001

    property iri: owlapy.iri.IRI
        Gets the IRI of this object.

        Returns
            The IRI of this object.

    property str
        Gets the string representation of this object

        Returns
            The IRI as string

    property reminder: str
        The reminder of the IRI

    is_owl_thing() → bool
        Determines if this expression is the built in class owl:Thing. This method does not determine if the class is
        equivalent to owl:Thing.

        Returns
            Thing.

        Return type
            True if this expression is owl

    is_owl_nothing() → bool
        Determines if this expression is the built in class owl:Nothing. This method does not determine if the class
        is equivalent to owl:Nothing.

    get_object_complement_of() → owlapy.class_expression.class_expression.OWLObjectComplementOf
        Gets the object complement of this class expression.

        Returns
            A class expression that is the complement of this class expression.

    get_nnf() → OWLClass
        Gets the negation normal form of the complement of this expression.

        Returns
            A expression that represents the NNF of the complement of this expression.

class owlapy.class_expression.OWLNaryBooleanClassExpression(
    operands: Iterable[owlapy.class_expression.class_expression.OWLClassExpression])
    Bases: owlapy.class_expression.class_expression.OWLBooleanClassExpression,
            owlapy.meta_classes.HasOperands[owlapy.class_expression.class_expression.
            OWLClassExpression]

    OWLNaryBooleanClassExpression.

```



```

__slots__ = ()

operands() → Iterable[owlapy.class_expression.class_expression.OWLClassExpression]
    Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

    Returns
        The operands.

__repr__()

__eq__(other)

__hash__()

class owlapy.class_expression.OWLObjectUnionOf(
    operands: Iterable[owlapy.class_expression.class_expression.OWLClassExpression])
    Bases: OWLNaryBooleanClassExpression
    A union class expression ObjectUnionOf( CE1 ... CEn ) contains all individuals that are instances of at least one
    class expression CEi for 1 ≤ i ≤ n. (https://www.w3.org/TR/owl2-syntax/#Union\_of\_Class\_Expressions)

    __slots__ = '_operands'

    type_index: Final = 3002

class owlapy.class_expression.OWLObjectIntersectionOf(
    operands: Iterable[owlapy.class_expression.class_expression.OWLClassExpression])
    Bases: OWLNaryBooleanClassExpression
    An intersection class expression ObjectIntersectionOf( CE1 ... CEn ) contains all individuals that are instances of
    all class expressions CEi for 1 ≤ i ≤ n. (https://www.w3.org/TR/owl2-syntax/#Intersection\_of\_Class\_Expressions)

    __slots__ = '_operands'

    type_index: Final = 3001

class owlapy.class_expression.OWLRestriction
    Bases: owlapy.class_expression.class_expression.OWLAnonymousClassExpression
    Represents an Object Property Restriction or Data Property Restriction in the OWL 2 specification.

    __slots__ = ()

    abstract get_property() → owlapy.owl_property.OWLPropertyExpression

    Returns
        Property being restricted.

    is_data_restriction() → bool
        Determines if this is a data restriction.

    Returns
        True if this is a data restriction.

    is_object_restriction() → bool
        Determines if this is an object restriction.

    Returns
        True if this is an object restriction.

```

```

class owlapy.class_expression.OWLQuantifiedRestriction
    Bases: Generic[_T], OWLRestriction, owlapy.meta_classes.HasFiller[_T]
    Represents a quantified restriction.

    Parameters
        _T – value type

    __slots__ = ()

class owlapy.class_expression.OWLQuantifiedObjectRestriction(
    filler: owlapy.class_expression.class_expression.OWLClassExpression)
    Bases: OWLQuantifiedRestriction[owlapy.class_expression.class_expression.OWLClassExpression], OWLObjectRestriction
    Represents a quantified object restriction.

    __slots__ = ()

    get_filler() → owlapy.class_expression.class_expression.OWLClassExpression
        Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of
        a data restriction this will be a constant (data value). For quantified restriction this will be a class expression
        or a data range.

    Returns
        the value

class owlapy.class_expression.OWLObjectRestriction
    Bases: OWLRestriction
    Represents an Object Property Restriction in the OWL 2 specification.

    __slots__ = ()

    is_object_restriction() → bool
        Determines if this is an object restriction.

    Returns
        True if this is an object restriction.

    abstract get_property() → owlapy.owl_property.OWLObjectPropertyExpression

    Returns
        Property being restricted.

class owlapy.class_expression.OWLHasValueRestriction(value: _T)
    Bases: Generic[_T], OWLRestriction, owlapy.meta_classes.HasFiller[_T]
    Represent a HasValue restriction in the OWL 2

    Parameters
        value – The value type _T.

    __slots__ = ()

    __eq__(other)

    __hash__()

```

**get\_filler()** → *\_T*

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**  
the value

**class** owlapy.class\_expression.OWLDataRestriction

Bases: *OWLRestriction*

Represents a Data Property Restriction.

**\_\_slots\_\_** = ()

**is\_data\_restriction()** → bool

Determines if this is a data restriction.

**Returns**  
True if this is a data restriction.

**class** owlapy.class\_expression.OWLCardinalityRestriction(*cardinality: int, filler: \_F*)

Bases: *Generic[\_F]*, *OWLQuantifiedRestriction[\_F]*, *owlapy.meta\_classes.HasCardinality*

Base interface for owl min and max cardinality restriction.

**Parameters**  
*\_F* – Type of filler.

**\_\_slots\_\_** = ()

**get\_cardinality()** → int

Gets the cardinality of a restriction.

**Returns**  
The cardinality. A non-negative integer.

**get\_filler()** → *\_F*

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**  
the value

**class** owlapy.class\_expression.OWLObjectCardinalityRestriction(*cardinality: int, property: owlapy.owl\_property.OWLObjectPropertyExpression, filler: owlapy.class\_expression.class\_expression.OWLClassExpression*)

Bases: *OWLCardinalityRestriction[owlapy.class\_expression.class\_expression.OWLClassExpression]*, *OWLQuantifiedObjectRestriction*

Represents Object Property Cardinality Restrictions in the OWL 2 specification.

**\_\_slots\_\_** = ()

**get\_property()** → *owlapy.owl\_property.OWLObjectPropertyExpression*

**Returns**  
Property being restricted.

```

__repr__()

__eq__(other)

__hash__()

class owlapy.class_expression.OWLObjectHasSelf(
    property: owlapy.owl_property.OWLObjectPropertyExpression)
    Bases: OWLObjectRestriction

    A self-restriction ObjectHasSelf( OPE ) consists of an object property expression OPE, and it contains all those
    individuals that are connected by OPE to themselves. (https://www.w3.org/TR/owl2-syntax/#Self-Restriction)

    __slots__ = '_property'

    type_index: Final = 3011

    get_property() → owlapy.owl_property.OWLObjectPropertyExpression

        Returns
        Property being restricted.

    __eq__(other)

    __hash__()

    __repr__()

class owlapy.class_expression.OWLDataOneOf(
    values: owlapy.owl_literal.OWLLiteral | Iterable[owlapy.owl_literal.OWLLiteral])
    Bases: owlapy.owl_data_ranges.OWLDataRange, owlapy.meta_classes.HasOperands[owlapy.
    owl_literal.OWLLiteral]

    An enumeration of literals DataOneOf( lt1 ... ltn ) contains exactly the explicitly specified literals lti with  $1 \leq i \leq n$ .
    The resulting data range has arity one. (https://www.w3.org/TR/owl2-syntax/#Enumeration\_of\_Literals)

    type_index: Final = 4003

    __repr__()

    __hash__()

    __eq__(other)

    values() → Iterable[owlapy.owl_literal.OWLLiteral]
        Gets the values that are in the oneOf.

        Returns
        The values of this { @code DataOneOf } class expression.

    operands() → Iterable[owlapy.owl_literal.OWLLiteral]
        Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

        Returns
        The operands.

class owlapy.class_expression.OWLQuantifiedDataRestriction(
    filler: owlapy.owl_data_ranges.OWLDataRange)
    Bases: OWLQuantifiedRestriction[owlapy.owl_data_ranges.OWLDataRange], OWLDataRestriction

    Represents a quantified data restriction.

```

`__slots__ = ()`

`get_filler()` → *owlapy.owl\_data\_ranges.OWLDataRange*

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**

the value

```
class owlapy.class_expression.OWLDataCardinalityRestriction (cardinality: int,
    property: owlapy.owl_property.OWLDataPropertyExpression,
    filler: owlapy.owl_data_ranges.OWLDataRange)
```

Bases: *OWLCardinalityRestriction*[*owlapy.owl\_data\_ranges.OWLDataRange*], *OWLQuantifiedDataRestriction*, *OWLDataRestriction*

Represents Data Property Cardinality Restrictions.

`__slots__ = ()`

`__hash__()`

`__repr__()`

`__eq__(other)`

`get_property()` → *owlapy.owl\_property.OWLDataPropertyExpression*

**Returns**

Property being restricted.

```
class owlapy.class_expression.OWLObjectSomeValuesFrom (
    property: owlapy.owl_property.OWLObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OWLClassExpression)
```

Bases: *OWLQuantifiedObjectRestriction*

An existential class expression *ObjectSomeValuesFrom*( OPE CE ) consists of an object property expression OPE and a class expression CE, and it contains all those individuals that are connected by OPE to an individual that is an instance of CE.

`__slots__ = ('_property', '_filler')`

`type_index: Final = 3005`

`__repr__()`

`__eq__(other)`

`__hash__()`

`get_property()` → *owlapy.owl\_property.OWLObjectPropertyExpression*

**Returns**

Property being restricted.

```
class owlapy.class_expression.OWLObjectAllValuesFrom (
    property: owlapy.owl_property.OWLObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OWLClassExpression)
```

Bases: *OWLQuantifiedObjectRestriction*

A universal class expression `ObjectAllValuesFrom( OPE CE )` consists of an object property expression `OPE` and a class expression `CE`, and it contains all those individuals that are connected by `OPE` only to individuals that are instances of `CE`. ([https://www.w3.org/TR/owl2-syntax/#Universal\\_Quantification](https://www.w3.org/TR/owl2-syntax/#Universal_Quantification))

```
__slots__ = ('_property', '_filler')
```

```
type_index: Final = 3006
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
get_property() → owlapy.owl_property.OWLObjectPropertyExpression
```

#### Returns

Property being restricted.

```
class owlapy.class_expression.OWLObjectHasValue(  
    property: owlapy.owl_property.OWLObjectPropertyExpression,  
    individual: owlapy.owl_individual.OWLIndividual)
```

Bases: *OWLHasValueRestriction[owlapy.owl\_individual.OWLIndividual]*, *OWLObjectRestriction*

A has-value class expression `ObjectHasValue( OPE a )` consists of an object property expression `OPE` and an individual `a`, and it contains all those individuals that are connected by `OPE` to `a`. Each such class expression can be seen as a syntactic shortcut for the class expression `ObjectSomeValuesFrom( OPE ObjectOneOf( a ) )`. ([https://www.w3.org/TR/owl2-syntax/#Individual\\_Value\\_Restriction](https://www.w3.org/TR/owl2-syntax/#Individual_Value_Restriction))

```
__slots__ = ('_property', '_v')
```

```
type_index: Final = 3007
```

```
get_property() → owlapy.owl_property.OWLObjectPropertyExpression
```

#### Returns

Property being restricted.

```
as_some_values_from() → owlapy.class_expression.class_expression.OWLClassExpression
```

A convenience method that obtains this restriction as an existential restriction with a nominal filler.

#### Returns

The existential equivalent of this value restriction.  $\text{simp}(\text{HasValue}(p\ a)) = \text{some}(p\ \{a\})$ .

```
__repr__()
```

```
class owlapy.class_expression.OWLDatatypeRestriction(  
    type_: owlapy.owl_datatype.OWLDatatype,  
    facet_restrictions: OWLFacetRestriction | Iterable[OWLFacetRestriction])
```

Bases: *owlapy.owl\_data\_ranges.OWLDataRange*

A datatype restriction `DatatypeRestriction( DT F1 lt1 ... Fn ltn )` consists of a unary datatype `DT` and `n` pairs  $(F_i, l_{ti})$ . The resulting data range is unary and is obtained by restricting the value space of `DT` according to the semantics of all  $(F_i, v_i)$  (multiple pairs are interpreted conjunctively), where  $v_i$  are the data values of the literals  $l_{ti}$ . ([https://www.w3.org/TR/owl2-syntax/#Datatype\\_Restrictions](https://www.w3.org/TR/owl2-syntax/#Datatype_Restrictions))

```

__slots__ = ('_type', '_facet_restrictions')

type_index: Final = 4006

get_datatype() → owlapy.owl_datatype.OWLDatatype

get_facet_restrictions() → Sequence[OWLFacetRestriction]

__eq__(other)

__hash__()

__repr__()

class owlapy.class_expression.OWLFacet (remainder: str, symbolic_form: str,
    operator: Callable[[_X, _X], bool])
    Bases: _Vocabulary, enum.Enum
    Enumerations for OWL facets.
    property symbolic_form
    property operator
    static from_str(name: str) → OWLFacet

    MIN_INCLUSIVE: Final
    MIN_EXCLUSIVE: Final
    MAX_INCLUSIVE: Final
    MAX_EXCLUSIVE: Final
    LENGTH: Final
    MIN_LENGTH: Final
    MAX_LENGTH: Final
    PATTERN: Final
    TOTAL_DIGITS: Final
    FRACTION_DIGITS: Final

class owlapy.class_expression.OWLFacetRestriction (facet: owlapy.vocab.OWLFacet,
    literal: Literals)
    Bases: owlapy.owl_object.OWLObject
    A facet restriction is used to restrict a particular datatype.
    __slots__ = ('_facet', '_literal')
    type_index: Final = 4007
    get_facet() → owlapy.vocab.OWLFacet
    get_facet_value() → owlapy.owl_literal.OWLLiteral
    __eq__(other)

```

```

__hash__()

__repr__()

class owlapy.class_expression.OwlObjectMinCardinality (cardinality: int,
    property: owlapy.owl_property.OwlObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OwlClassExpression)
    Bases: OwlObjectCardinalityRestriction

    A minimum cardinality expression ObjectMinCardinality( n OPE CE ) consists of a nonnegative integer n, an object
    property expression OPE, and a class expression CE, and it contains all those individuals that are connected by
    OPE to at least n different individuals that are instances of CE. (https://www.w3.org/TR/owl2-syntax/#Minimum\_Cardinality)

    __slots__ = ('_cardinality', '_filler', '_property')

    type_index: Final = 3008

class owlapy.class_expression.OwlObjectMaxCardinality (cardinality: int,
    property: owlapy.owl_property.OwlObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OwlClassExpression)
    Bases: OwlObjectCardinalityRestriction

    A maximum cardinality expression ObjectMaxCardinality( n OPE CE ) consists of a nonnegative integer n, an
    object property expression OPE, and a class expression CE, and it contains all those individuals that are connected
    by OPE

    to at most n different individuals that are instances of CE. (https://www.w3.org/TR/owl2-syntax/
    #Maximum\_Cardinality)

    __slots__ = ('_cardinality', '_filler', '_property')

    type_index: Final = 3010

class owlapy.class_expression.OwlObjectExactCardinality (cardinality: int,
    property: owlapy.owl_property.OwlObjectPropertyExpression,
    filler: owlapy.class_expression.class_expression.OwlClassExpression)
    Bases: OwlObjectCardinalityRestriction

    An exact cardinality expression ObjectExactCardinality( n OPE CE ) consists of a nonnegative integer n,
    an object
    property expression OPE, and a class expression CE, and it contains all those individuals that are connected
    by to exactly n different individuals that are instances of CE.

    (https://www.w3.org/TR/owl2-syntax/#Exact\_Cardinality)

    __slots__ = ('_cardinality', '_filler', '_property')

    type_index: Final = 3009

    as_intersection_of_min_max()
        → owlapy.class_expression.nary_boolean_expression.OwlObjectIntersectionOf

    Obtains an equivalent form that is a conjunction of a min cardinality and max cardinality restriction.

    Returns
        The semantically equivalent but structurally simpler form ( $= 1 \text{ R C} \Rightarrow 1 \text{ R C}$  and  $\leq 1 \text{ R C}$ ).

class owlapy.class_expression.OwlDataSomeValuesFrom (
    property: owlapy.owl_property.OwlDataPropertyExpression,
    filler: owlapy.owl_data_ranges.OwlDataRange)

```



Bases: *OWLQuantifiedDataRestriction*

An existential class expression `DataSomeValuesFrom( DPE1 ... DPEn DR )` consists of  $n$  data property expressions  $DPE_i$ ,  $1 \leq i \leq n$ , and a data range  $DR$  whose arity must be  $n$ . Such a class expression contains all those individuals that are connected by  $DPE_i$  to literals  $l_{ti}$ ,  $1 \leq i \leq n$ , such that the tuple  $( l_{t1} , \dots , l_{tn} )$  is in  $DR$ . A class expression of the form `DataSomeValuesFrom( DPE DR )` can be seen as a syntactic shortcut for the class expression `DataMinCardinality( 1 DPE DR )`. ([https://www.w3.org/TR/owl2-syntax/#Existential\\_Quantification\\_2](https://www.w3.org/TR/owl2-syntax/#Existential_Quantification_2))

```
__slots__ = '_property'
```

```
type_index: Final = 3012
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
get_property() → owlapy.owl_property.OWLDataPropertyExpression
```

#### Returns

Property being restricted.

```
class owlapy.class_expression.OWLDataAllValuesFrom(  
    property: owlapy.owl_property.OWLDataPropertyExpression,  
    filler: owlapy.owl_data_ranges.OWLDataRange)
```

Bases: *OWLQuantifiedDataRestriction*

A universal class expression `DataAllValuesFrom( DPE1 ... DPEn DR )` consists of  $n$  data property expressions  $DPE_i$ ,  $1 \leq i \leq n$ , and a data range  $DR$  whose arity must be  $n$ . Such a class expression contains all those individuals that

**are connected by  $DPE_i$  only to literals  $l_{ti}$ ,  $1 \leq i \leq n$ , such that each tuple  $( l_{t1} , \dots , l_{tn} )$  is in  $DR$ .**

#### A class

expression of the form `DataAllValuesFrom( DPE DR )` can be seen as a syntactic shortcut for the class expression `DataMaxCardinality( 0 DPE DataComplementOf( DR ) )`. ([https://www.w3.org/TR/owl2-syntax/#Universal\\_Quantification\\_2](https://www.w3.org/TR/owl2-syntax/#Universal_Quantification_2))

```
__slots__ = '_property'
```

```
type_index: Final = 3013
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
get_property() → owlapy.owl_property.OWLDataPropertyExpression
```

#### Returns

Property being restricted.

```
class owlapy.class_expression.OWLDataHasValue(  
    property: owlapy.owl_property.OWLDataPropertyExpression,  
    value: owlapy.owl_literal.OWLLiteral)
```

Bases: *OWLHasValueRestriction*[*owlapy.owl\_literal.OWLLiteral*], *OWLDataRestriction*

A has-value class expression `DataHasValue( DPE lt )` consists of a data property expression  $DPE$  and a literal  $lt$ , and it contains all those individuals that are connected by  $DPE$  to  $lt$ . Each such class expression can be seen as a

syntactic shortcut for the class expression `DataSomeValuesFrom( DPE DataOneOf( It ) )`. ([https://www.w3.org/TR/owl2-syntax/#Literal\\_Value\\_Restriction](https://www.w3.org/TR/owl2-syntax/#Literal_Value_Restriction))

```
__slots__ = '_property'
```

```
type_index: Final = 3014
```

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
as_some_values_from() → owlapy.class_expression.class_expression.OWLClassExpression
```

A convenience method that obtains this restriction as an existential restriction with a nominal filler.

#### Returns

The existential equivalent of this value restriction.  $\text{simp}(\text{HasValue}(p\ a)) = \text{some}(p\ \{a\})$ .

```
get_property() → owlapy.owl_property.OWLDataPropertyExpression
```

#### Returns

Property being restricted.

```
class owlapy.class_expression.OWLDataMinCardinality(cardinality: int,  

property: owlapy.owl_property.OWLDataPropertyExpression,  

filler: owlapy.owl_data_ranges.OWLDataRange)
```

Bases: *OWLDataCardinalityRestriction*

A minimum cardinality expression `DataMinCardinality( n DPE DR )` consists of a nonnegative integer *n*, a data property expression *DPE*, and a unary data range *DR*, and it contains all those individuals that are connected by *DPE* to at least *n* different literals in *DR*. ([https://www.w3.org/TR/owl2-syntax/#Minimum\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Minimum_Cardinality))

```
__slots__ = ('_cardinality', '_filler', '_property')
```

```
type_index: Final = 3015
```

```
class owlapy.class_expression.OWLDataMaxCardinality(cardinality: int,  

property: owlapy.owl_property.OWLDataPropertyExpression,  

filler: owlapy.owl_data_ranges.OWLDataRange)
```

Bases: *OWLDataCardinalityRestriction*

A maximum cardinality expression `ObjectMaxCardinality( n OPE CE )` consists of a nonnegative integer *n*, an object property expression *OPE*, and a class expression *CE*, and it contains all those individuals that are connected by *OPE* to at most *n* different individuals that are instances of *CE*. ([https://www.w3.org/TR/owl2-syntax/#Maximum\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Maximum_Cardinality))

```
__slots__ = ('_cardinality', '_filler', '_property')
```

```
type_index: Final = 3017
```

```
class owlapy.class_expression.OWLDataExactCardinality(cardinality: int,  

property: owlapy.owl_property.OWLDataPropertyExpression,  

filler: owlapy.owl_data_ranges.OWLDataRange)
```

Bases: *OWLDataCardinalityRestriction*

An exact cardinality expression `ObjectExactCardinality( n OPE CE )` consists of a nonnegative integer *n*, an object property expression *OPE*, and a class expression *CE*, and it contains all those individuals that are connected

by *OPE* to exactly *n* different individuals that are instances of *CE* ([https://www.w3.org/TR/owl2-syntax/#Exact\\_Cardinality](https://www.w3.org/TR/owl2-syntax/#Exact_Cardinality))

```
__slots__ = ('_cardinality', '_filler', '_property')
```

```
type_index: Final = 3016
```

```
as_intersection_of_min_max()
```

→ *owlapy.class\_expression.nary\_boolean\_expression.OwlObjectIntersectionOf*

Obtains an equivalent form that is a conjunction of a min cardinality and max cardinality restriction.

#### Returns

The semantically equivalent but structurally simpler form ( $= 1 \text{ R } D$ )  $= \geq 1 \text{ R } D$  and  $\leq 1 \text{ R } D$ .

```
class owlapy.class_expression.OwlObjectOneOf(
```

```
    values: owlapy.owl_individual.OwlIndividual | Iterable[owlapy.owl_individual.OwlIndividual])
```

Bases: *owlapy.class\_expression.class\_expression.OwlAnonymousClassExpression*, *owlapy.meta\_classes.HasOperands[owlapy.owl\_individual.OwlIndividual]*

An enumeration of individuals *ObjectOneOf*( $a_1 \dots a_n$ ) contains exactly the individuals  $a_i$  with  $1 \leq i \leq n$ . ([https://www.w3.org/TR/owl2-syntax/#Enumeration\\_of\\_Individuals](https://www.w3.org/TR/owl2-syntax/#Enumeration_of_Individuals))

```
__slots__ = '_values'
```

```
type_index: Final = 3004
```

```
individuals() → Iterable[owlapy.owl_individual.OwlIndividual]
```

Gets the individuals that are in the oneOf. These individuals represent the exact instances (extension) of this class expression.

#### Returns

The individuals that are the values of this {`@code ObjectOneOf`} class expression.

```
operands() → Iterable[owlapy.owl_individual.OwlIndividual]
```

Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

#### Returns

The operands.

```
as_object_union_of() → owlapy.class_expression.class_expression.OwlClassExpression
```

Simplifies this enumeration to a union of singleton nominals.

#### Returns

This enumeration in a more standard DL form.  $\text{simp}(\{a\}) = \{a\}$   $\text{simp}(\{a_0, \dots, \{a_n\}) = \text{unionOf}(\{a_0\}, \dots, \{a_n\})$

```
__hash__()
```

```
__eq__(other)
```

```
__repr__()
```

## owlapy.converter

Format converter.

## Attributes

---

*converter*

---

## Classes

<code>VariablesMapping</code>	Helper class for owl-to-sparql conversion.
<code>Owl2SparqlConverter</code>	Convert owl (owlapy model class expressions) to SPARQL.

## Functions

<code>peek(x)</code>	Peek the last element of an array.
<code>owl_expression_to_sparql(→ str)</code>	Convert an OWL Class Expression ( <a href="https://www.w3.org/TR/owl2-syntax/#Class_Expressions">https://www.w3.org/TR/owl2-syntax/#Class_Expressions</a> ) into a SPARQL query
<code>owl_expression_to_sparql_with_confusion_map(→ str)</code>	Convert an OWL Class Expression ( <a href="https://www.w3.org/TR/owl2-syntax/#Class_Expressions">https://www.w3.org/TR/owl2-syntax/#Class_Expressions</a> ) into a SPARQL query

## Module Contents

`owlapy.converter.peek(x)`

Peek the last element of an array.

### Returns

The last element `arr[-1]`.

`class owlapy.converter.VariablesMapping`

Helper class for owl-to-sparql conversion.

`__slots__ = ('class_cnt', 'prop_cnt', 'ind_cnt', 'dict')`

`class_cnt = 0`

`prop_cnt = 0`

`ind_cnt = 0`

`dict`

`get_variable(e: owlapy.owl_object.OWLEntity) → str`

`new_individual_variable() → str`

`new_property_variable() → str`

`__contains__(item: owlapy.owl_object.OWLEntity) → bool`

`__getitem__(item: owlapy.owl_object.OWLEntity) → str`

`class owlapy.converter.Owl2SparqlConverter`

Convert owl (owlapy model class expressions) to SPARQL.

`__slots__ = ('ce', 'sparql', 'variables', 'parent', 'parent_var', 'properties', 'variable_entities', 'cnt', ...)`

`ce: owlapy.class_expression.OWLClassExpression`

```

sparql: List[str]

variables: List[str]

parent: List[owlapy.class_expression.OWLClassExpression]

parent_var: List[str]

variable_entities: Set[owlapy.owl_object.OWLEntity]

properties: Dict[int, List[owlapy.owl_object.OWLEntity]]

mapping: VariablesMapping

grouping_vars: Dict[owlapy.class_expression.OWLClassExpression, Set[str]]

having_conditions: Dict[owlapy.class_expression.OWLClassExpression, Set[str]]

cnt: int

for_all_de_morgan: bool

named_individuals: bool

```

```

convert (root_variable: str, ce: owlapy.class_expression.OWLClassExpression,
        for_all_de_morgan: bool = True, named_individuals: bool = False)

```

Used to convert owl class expression to SPARQL syntax.

#### Parameters

- **root\_variable** (*str*) – Root variable name that will be used in SPARQL query.
- **ce** (*OWLClassExpression*) – The owl class expression to convert.
- **named\_individuals** (*bool*) – If ‘True’ return only entities that are instances of owl:NamedIndividual.

#### Returns

The SPARQL query.

#### Return type

list[str]

```
property modal_depth
```

```
abstract render(e)
```

```
stack_variable(var)
```

```
stack_parent(parent: owlapy.class_expression.OWLClassExpression)
```

```
property current_variable
```

```
abstract process(ce: owlapy.class_expression.OWLClassExpression)
```

```
forAll(ce: owlapy.class_expression.OWLObjectAllValuesFrom)
```

```
forAllDeMorgan(ce: owlapy.class_expression.OWLObjectAllValuesFrom)
```

```
new_count_var() → str
```

```
append_triple(subject, predicate, object_)
```

**append** (*frag*)

**triple** (*subject*, *predicate*, *object\_*)

**as\_query** (*root\_variable*: str, *ce*: *owlapy.class\_expression.OWLClassExpression*,  
    *for\_all\_de\_morgan*: bool = True, *count*: bool = False,  
    *values*: Iterable[*owlapy.owl\_individual.OWLNamedIndividual*] | None = None,  
    *named\_individuals*: bool = False) → str

**as\_confusion\_matrix\_query** (*root\_variable*: str, *ce*: *owlapy.class\_expression.OWLClassExpression*,  
    *positive\_examples*: Iterable[*owlapy.owl\_individual.OWLNamedIndividual*],  
    *negative\_examples*: Iterable[*owlapy.owl\_individual.OWLNamedIndividual*],  
    *for\_all\_de\_morgan*: bool = True, *named\_individuals*: bool = False) → str

*owlapy.converter.converter*

*owlapy.converter.owl\_expression\_to\_sparql* (  
    *expression*: *owlapy.class\_expression.OWLClassExpression* = None, *root\_variable*: str = '?x',  
    *values*: Iterable[*owlapy.owl\_individual.OWLNamedIndividual*] | None = None,  
    *for\_all\_de\_morgan*: bool = True, *named\_individuals*: bool = False) → str

Convert an OWL Class Expression ([https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions)) into a SPARQL query root variable: the variable that will be projected expression: the class expression to be transformed to a SPARQL query

values: positive or negative examples from a class expression problem. Unclear for\_all\_de\_morgan: if set to True, the SPARQL mapping will use the mapping containing the nested FILTER NOT EXISTS patterns for the universal quantifier ( $\neg(\exists r. \neg C)$ ), instead of the counting query named\_individuals: if set to True, the generated SPARQL query will return only entities that are instances of owl:NamedIndividual

*owlapy.converter.owl\_expression\_to\_sparql\_with\_confusion\_matrix* (  
    *expression*: *owlapy.class\_expression.OWLClassExpression*,  
    *positive\_examples*: Iterable[*owlapy.owl\_individual.OWLNamedIndividual*] | None,  
    *negative\_examples*: Iterable[*owlapy.owl\_individual.OWLNamedIndividual*] | None,  
    *root\_variable*: str = '?x', *for\_all\_de\_morgan*: bool = True, *named\_individuals*: bool = False) → str

Convert an OWL Class Expression ([https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions)) into a SPARQL query root variable: the variable that will be projected expression: the class expression to be transformed to a SPARQL query positive\_examples: positive examples from a class expression problem negative\_examples: positive examples from a class expression problem for\_all\_de\_morgan: if set to True, the SPARQL mapping will use the mapping containing the nested FILTER NOT EXISTS patterns for the universal quantifier ( $\neg(\exists r. \neg C)$ ), instead of the counting query named\_individuals: if set to True, the generated SPARQL query will return only entities that are instances of owl:NamedIndividual

## owlapy.entities

Entities are the fundamental building blocks of OWL 2 ontologies, and they define the vocabulary — the named terms — of an ontology. In logic, the set of entities is usually said to constitute the signature of an ontology.

Classes, datatypes, object properties, data properties, annotation properties, and named individuals are entities, and they are all uniquely identified by an IR.

## owlapy.iri

OWL IRI

## Classes

<i>IRI</i>	An IRI, consisting of a namespace and a remainder.
------------	--

## Module Contents

**class** owlapy.iri.IRI (namespace: str | owlapy.namespaces.Namespaces, remainder: str = "")

Bases: owlapy.owl\_annotation.OWLAnnotationSubject, owlapy.owl\_annotation.OWLAnnotationValue

An IRI, consisting of a namespace and a remainder.

**\_\_slots\_\_** = ('\_namespace', '\_remainder', '\_\_weakref\_\_')

**type\_index**: Final = 0

**static create** (iri: str | owlapy.namespaces.Namespaces, remainder: str = None) → IRI

**\_\_repr\_\_**()

**\_\_eq\_\_** (other)

**\_\_hash\_\_**()

**is\_nothing**()

Determines if this IRI is equal to the IRI that owl:Nothing is named with.

### Returns

True if this IRI is equal to <http://www.w3.org/2002/07/owl#Nothing> and otherwise False.

**is\_thing**()

Determines if this IRI is equal to the IRI that owl:Thing is named with.

### Returns

True if this IRI is equal to <http://www.w3.org/2002/07/owl#Thing> and otherwise False.

**is\_reserved\_vocabulary**() → bool

Determines if this IRI is in the reserved vocabulary. An IRI is in the reserved vocabulary if it starts with <http://www.w3.org/1999/02/22-rdf-syntax-ns#> or <http://www.w3.org/2000/01/rdf-schema#> or <http://www.w3.org/2001/XMLSchema#> or <http://www.w3.org/2002/07/owl#>.

### Returns

True if the IRI is in the reserved vocabulary, otherwise False.

**as\_iri**() → IRI

### Returns

if the value is an IRI, return it. Return None otherwise.

**as\_str**() → str

CD: Should be deprecated. :returns: The string that specifies the IRI.

**property str**: str

Returns: The string that specifies the IRI.

**property reminder**: str

Returns: The string corresponding to the reminder of the IRI.

`get_namespace()` → str

**Returns**

The namespace as string.

`get_remainder()` → str

**Returns**

The remainder (coincident with NCName usually) for this IRI.

## owlapy.meta\_classes

Meta classes for OWL objects.

### Classes

<i>HasIRI</i>	Simple class to access the IRI.
<i>HasOperands</i>	An interface to objects that have a collection of operands.
<i>HasFiller</i>	An interface to objects that have a filler.
<i>HasCardinality</i>	An interface to objects that have a cardinality.

### Module Contents

**class** owlapy.meta\_classes.**HasIRI**

Simple class to access the IRI.

**\_\_slots\_\_** = ()

**abstract property** iri

Gets the IRI of this object.

**Returns**

The IRI of this object.

**property** str: str

**Abstractmethod**

Gets the string representation of this object

**Returns**

The IRI as string

**class** owlapy.meta\_classes.**HasOperands**

Bases: Generic[\_T]

An interface to objects that have a collection of operands.

**Parameters**

**\_T** – Operand type.

**\_\_slots\_\_** = ()

**abstract** operands() → Iterable[\_T]

Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

**Returns**

The operands.



```
class owlapy.meta_classes.HasFiller
```

```
    Bases: Generic[_T]
```

An interface to objects that have a filler.

**Parameters**

**\_T** – Filler type.

```
    __slots__ = ()
```

```
    abstract get_filler() → _T
```

Gets the filler for this restriction. In the case of an object restriction this will be an individual, in the case of a data restriction this will be a constant (data value). For quantified restriction this will be a class expression or a data range.

**Returns**

    the value

```
class owlapy.meta_classes.HasCardinality
```

An interface to objects that have a cardinality.

```
    __slots__ = ()
```

```
    abstract get_cardinality() → int
```

Gets the cardinality of a restriction.

**Returns**

    The cardinality. A non-negative integer.

## owlapy.namespaces

Namespaces.

### Attributes

---

*OWL*

*RDFS*

*RDF*

*XSD*

---

### Classes

---

*Namespaces*

Namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup

---

## Module Contents

```
class owlapy.namespaces.Namespaces (prefix: str, ns: str)
```

Namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references

```

__slots__ = ('_prefix', '_ns')

property ns: str

property prefix: str

__repr__()

__hash__()

__eq__(other)

```

```

owlapy.namespaces.OWL: Final

owlapy.namespaces.RDFS: Final

owlapy.namespaces.RDF: Final

owlapy.namespaces.XSD: Final

```

## owlapy.owl\_annotation

OWL Annotations

### Classes

<i>OWLAnnotationObject</i>	A marker interface for the values (objects) of annotations.
<i>OWLAnnotationSubject</i>	A marker interface for annotation subjects, which can either be IRIs or anonymous individuals
<i>OWLAnnotationValue</i>	A marker interface for annotation values, which can either be an IRI (URI), Literal or Anonymous Individual.

### Module Contents

**class** owlapy.owl\_annotation.**OWLAnnotationObject**

Bases: *owlapy.owl\_object.OWLObject*

A marker interface for the values (objects) of annotations.

```
__slots__ = ()
```

```
as_iri()
```

#### Returns

if the value is an IRI, return it. Return None otherwise.

```
as_anonymous_individual()
```

#### Returns

if the value is an anonymous, return it. Return None otherwise.

**class** owlapy.owl\_annotation.**OWLAnnotationSubject**

Bases: *OWLAnnotationObject*

A marker interface for annotation subjects, which can either be IRIs or anonymous individuals

```
__slots__ = ()
```

**class** owlapy.owl\_annotation.OWLAnnotationValue

Bases: *OWLAnnotationObject*

A marker interface for annotation values, which can either be an IRI (URI), Literal or Anonymous Individual.

**\_\_slots\_\_** = ()

**is\_literal**() → bool

**Returns**

true if the annotation value is a literal

**as\_literal**()

**Returns**

if the value is a literal, returns it. Return None otherwise

## owlapy.owl\_axiom

OWL Axioms

## Classes

<i>OWLAxiom</i>	Represents Axioms in the OWL 2 Specification.
<i>OWLLogicalAxiom</i>	A base interface of all axioms that affect the logical meaning of an ontology. This excludes declaration
<i>OWLPropertyAxiom</i>	The base interface for property axioms.
<i>OWLObjectPropertyAxiom</i>	The base interface for object property axioms.
<i>OWLDatapropertyAxiom</i>	The base interface for data property axioms.
<i>OWLIndividualAxiom</i>	The base interface for individual axioms.
<i>OWLClassAxiom</i>	The base interface for class axioms.
<i>OWLDeclarationAxiom</i>	Represents a Declaration axiom in the OWL 2 Specification. A declaration axiom declares an entity in an ontology.
<i>OWLDatatypeDefinitionAxiom</i>	A datatype definition <i>DatatypeDefinition</i> ( DT DR ) defines a new datatype DT as being semantically
<i>OWLHasKeyAxiom</i>	A key axiom <i>HasKey</i> ( CE ( OPE1 ... OPEm ) ( DPE1 ... DPEn ) ) states that each
<i>OWLNaryAxiom</i>	Represents an axiom that contains two or more operands that could also be represented with multiple pairwise
<i>OWLNaryClassAxiom</i>	Represents an axiom that contains two or more operands that could also be represented with
<i>OWLEquivalentClassesAxiom</i>	An equivalent classes axiom <i>EquivalentClasses</i> ( CE1 ... CEn ) states that all of the class expressions CEi,
<i>OWLDisjointClassesAxiom</i>	A disjoint classes axiom <i>DisjointClasses</i> ( CE1 ... CEn ) states that all of the class expressions CEi, 1 ≤ i ≤ n,
<i>OWLNaryIndividualAxiom</i>	Represents an axiom that contains two or more operands that could also be represented with
<i>OWLDifferentIndividualsAxiom</i>	An individual inequality axiom <i>DifferentIndividuals</i> ( a1 ... an ) states that all of the individuals ai,
<i>OWLSameIndividualAxiom</i>	An individual equality axiom <i>SameIndividual</i> ( a1 ... an ) states that all of the individuals ai, 1 ≤ i ≤ n,
<i>OWLNaryPropertyAxiom</i>	Represents an axiom that contains two or more operands that could also be represented with

continues on next page

Table 2 – continued from previous page

<i>OWLEquivalentObjectPropertiesAxiom</i>	An equivalent object properties axiom <i>EquivalentObjectProperties</i> ( OPE1 ... OPE <sub>n</sub> ) states that all of the object
<i>OWLDisjointObjectPropertiesAxiom</i>	A disjoint object properties axiom <i>DisjointObjectProperties</i> ( OPE1 ... OPE <sub>n</sub> ) states that all of the object
<i>OWLInverseObjectPropertiesAxiom</i>	An inverse object properties axiom <i>InverseObjectProperties</i> ( OPE1 OPE2 ) states that the object property
<i>OWLEquivalentDataPropertiesAxiom</i>	An equivalent data properties axiom <i>EquivalentDataProperties</i> ( DPE1 ... DPE <sub>n</sub> ) states that all the data property
<i>OWLDisjointDataPropertiesAxiom</i>	A disjoint data properties axiom <i>DisjointDataProperties</i> ( DPE1 ... DPE <sub>n</sub> ) states that all of the data property
<i>OWLSubClassOfAxiom</i>	A subclass axiom <i>SubClassOf</i> ( CE1 CE2 ) states that the class expression CE1 is a subclass of the class
<i>OWLDisjointUnionAxiom</i>	A disjoint union axiom <i>DisjointUnion</i> ( C CE1 ... CE <sub>n</sub> ) states that a class C is a disjoint union of the class
<i>OWLClassAssertionAxiom</i>	A class assertion <i>ClassAssertion</i> ( CE a ) states that the individual a is an instance of the class expression CE.
<i>OWLAnnotationProperty</i>	Represents an <i>AnnotationProperty</i> in the OWL 2 specification.
<i>OWLAnnotation</i>	Annotations are used in the various types of annotation axioms, which bind annotations to their subjects
<i>OWLAnnotationAxiom</i>	A super interface for annotation axioms.
<i>OWLAnnotationAssertionAxiom</i>	An annotation assertion <i>AnnotationAssertion</i> ( AP as av ) states that the annotation subject as — an IRI or an
<i>OWLSubAnnotationPropertyOfAxiom</i>	An annotation subproperty axiom <i>SubAnnotationPropertyOf</i> ( AP1 AP2 ) states that the annotation property AP1 is
<i>OWLAnnotationPropertyDomainAxiom</i>	An annotation property domain axiom <i>AnnotationPropertyDomain</i> ( AP U ) states that the domain of the annotation
<i>OWLAnnotationPropertyRangeAxiom</i>	An annotation property range axiom <i>AnnotationPropertyRange</i> ( AP U )
<i>OWLSubPropertyAxiom</i>	Base interface for object and data sub-property axioms.
<i>OWLSubObjectPropertyOfAxiom</i>	Object subproperty axioms are analogous to subclass axioms, and they come in two forms.
<i>OWLSubDataPropertyOfAxiom</i>	A data subproperty axiom <i>SubDataPropertyOf</i> ( DPE1 DPE2 ) states that the data property expression DPE1 is a
<i>OWLPropertyAssertionAxiom</i>	Base class for Property Assertion axioms.
<i>OWLObjectPropertyAssertionAxiom</i>	A positive object property assertion <i>ObjectPropertyAssertion</i> ( OPE a1 a2 ) states that the individual a1 is
<i>OWLNegativeObjectPropertyAssertionAxiom</i>	A negative object property assertion <i>NegativeObjectPropertyAssertion</i> ( OPE a1 a2 ) states that the individual a1
<i>OWLDataPropertyAssertionAxiom</i>	A positive data property assertion <i>DataPropertyAssertion</i> ( DPE a lt ) states that the individual a is connected
<i>OWLNegativeDataPropertyAssertionAxiom</i>	A negative data property assertion <i>NegativeDataPropertyAssertion</i> ( DPE a lt ) states that the individual a is not
<i>OWLUnaryPropertyAxiom</i>	Base class for Unary property axiom.
<i>OWLObjectPropertyCharacteristicAxiom</i>	Base interface for functional object property axiom.

continues on next page

Table 2 – continued from previous page

<i>OWLFunctionalObjectPropertyAxiom</i>	An object property functionality axiom <code>FunctionalObjectProperty( OPE )</code> states that
<i>OWLASymmetricObjectPropertyAxiom</i>	An object property asymmetry axiom <code>AsymmetricObjectProperty( OPE )</code> states that
<i>OWLInverseFunctionalObjectPropertyAxiom</i>	An object property inverse functionality axiom <code>InverseFunctionalObjectProperty( OPE )</code>
<i>OWLIrreflexiveObjectPropertyAxiom</i>	An object property irreflexivity axiom <code>IrreflexiveObjectProperty( OPE )</code> states that the
<i>OWLReflexiveObjectPropertyAxiom</i>	An object property reflexivity axiom <code>ReflexiveObjectProperty( OPE )</code> states that the
<i>OWLSymmetricObjectPropertyAxiom</i>	An object property symmetry axiom <code>SymmetricObjectProperty( OPE )</code> states that
<i>OWLTransitiveObjectPropertyAxiom</i>	An object property transitivity axiom <code>TransitiveObjectProperty( OPE )</code> states that the
<i>OWLDataPropertyCharacteristicAxiom</i>	Base interface for Functional data property axiom.
<i>OWLFunctionalDataPropertyAxiom</i>	A data property functionality axiom <code>FunctionalDataProperty( DPE )</code> states that
<i>OWLPropertyDomainAxiom</i>	Base class for Property Domain axioms.
<i>OWLPropertyRangeAxiom</i>	Base class for Property Range axioms.
<i>OWLObjectPropertyDomainAxiom</i>	An object property domain axiom <code>ObjectPropertyDomain( OPE CE )</code> states that the domain of the
<i>OWLDataPropertyDomainAxiom</i>	A data property domain axiom <code>DataPropertyDomain( DPE CE )</code> states that the domain of the
<i>OWLObjectPropertyRangeAxiom</i>	An object property range axiom <code>ObjectPropertyRange( OPE CE )</code> states that the range of the object property
<i>OWLDataPropertyRangeAxiom</i>	A data property range axiom <code>DataPropertyRange( DPE DR )</code> states that the range of the data property

## Module Contents

**class** owlapy.owl\_axiom.**OWLAxiom** (*annotations: Iterable[OWLAnnotation] | None = None*)

Bases: owlapy.owl\_object.OWLObject

Represents Axioms in the OWL 2 Specification.

An OWL ontology contains a set of axioms. These axioms can be annotation axioms, declaration axioms, imports axioms or logical axioms.

**\_\_slots\_\_** = '\_annotations'

**annotations** () → List[OWLAnnotation] | None

**is\_annotated** () → bool

**is\_logical\_axiom** () → bool

**is\_annotation\_axiom** () → bool

**class** owlapy.owl\_axiom.**OWLLogicalAxiom** (*annotations: Iterable[OWLAnnotation] | None = None*)

Bases: OWLAxiom

A base interface of all axioms that affect the logical meaning of an ontology. This excludes declaration axioms (including imports declarations) and annotation axioms.

**\_\_slots\_\_** = ()

```

    is_logical_axiom() → bool

class owlapy.owl_axiom.OWLPropertyAxiom(annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLLogicalAxiom
    The base interface for property axioms.
    __slots__ = ()

class owlapy.owl_axiom.OWLObjectPropertyAxiom(
    annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLPropertyAxiom
    The base interface for object property axioms.
    __slots__ = ()

class owlapy.owl_axiom.OWLDataPropertyAxiom(
    annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLPropertyAxiom
    The base interface for data property axioms.
    __slots__ = ()

class owlapy.owl_axiom.OWLIndividualAxiom(annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLLogicalAxiom
    The base interface for individual axioms.
    __slots__ = ()

class owlapy.owl_axiom.OWLClassAxiom(annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLLogicalAxiom
    The base interface for class axioms.
    __slots__ = ()

class owlapy.owl_axiom.OWLDeclarationAxiom(entity: owlapy.owl_object.OWLEntity,
    annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLAxiom
    Represents a Declaration axiom in the OWL 2 Specification. A declaration axiom declares an entity in an ontology.
    It doesn't affect the logical meaning of the ontology.
    __slots__ = '_entity'
    get_entity() → owlapy.owl_object.OWLEntity
    __eq__(other)
    __hash__()
    __repr__()

class owlapy.owl_axiom.OWLDatatypeDefinitionAxiom(
    datatype: owlapy.owl_datatype.OWLDatatype, datarange: owlapy.owl_datatype.OWLDataRange,
    annotations: Iterable[OWLAnnotation] | None = None)

```

Bases: *OWLLogicalAxiom*

A datatype definition `DatatypeDefinition( DT DR )` defines a new datatype `DT` as being semantically equivalent to the data range `DR`; the latter must be a unary data range. This axiom allows one to use the defined datatype `DT` as a synonym for `DR` — that is, in any expression in the ontology containing such an axiom, `DT` can be replaced with `DR` without affecting the meaning of the ontology.

([https://www.w3.org/TR/owl2-syntax/#Datatype\\_Definitions](https://www.w3.org/TR/owl2-syntax/#Datatype_Definitions))

```
__slots__ = ('_datatype', '_datarange')

get_datatype() → owlapy.owl_datatype.OWLDataType

get_datarange() → owlapy.owl_datatype.OWLDataRange

__eq__(other)

__hash__()

__repr__()
```

```
class owlapy.owl_axiom.OWLHasKeyAxiom(
    class_expression: owlapy.class_expression.OWLClassExpression,
    property_expressions: List[owlapy.owl_property.OWLPropertyExpression],
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLLogicalAxiom*, *owlapy.meta\_classes.HasOperands[owlapy.owl\_property.OWLPropertyExpression]*

A key axiom `HasKey( CE ( OPE1 ... OPEm ) ( DPE1 ... DPEn ) )` states that each (named) instance of the class expression `CE` is uniquely identified by the object property expressions `OPEi` and/or the data property expressions `DPEj` — that is, no two distinct (named) instances of `CE` can coincide on the values of all object property expressions `OPEi` and all data property expressions `DPEj`. In each such axiom in an OWL ontology, `m` or `n` (or both) must be larger than zero. A key axiom of the form `HasKey( owl:Thing ( OPE ) )` is similar to the axiom `InverseFunctionalObjectProperty( OPE )`, the main differences being that the former axiom is applicable only to individuals that are explicitly named in an ontology, while the latter axiom is also applicable to anonymous individuals and individuals whose existence is implied by existential quantification.

(<https://www.w3.org/TR/owl2-syntax/#Keys>)

```
__slots__ = ('_class_expression', '_property_expressions')

get_class_expression() → owlapy.class_expression.OWLClassExpression

get_property_expressions() → List[owlapy.owl_property.OWLPropertyExpression]

operands() → Iterable[owlapy.owl_property.OWLPropertyExpression]
```

Gets the operands - e.g., the individuals in a `sameAs` axiom, or the classes in an `equivalentClasses` axiom.

#### Returns

The operands.

```
__eq__(other)

__hash__()

__repr__()
```

```
class owlapy.owl_axiom.OwLNaryAxiom (annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: Generic[\_C], OWLAxiom

Represents an axiom that contains two or more operands that could also be represented with multiple pairwise axioms.

#### Parameters

**\_C** – Class of contained objects.

```
__slots__ = ()
```

```
abstract as_pairwise_axioms () → Iterable[OwLNaryAxiom[_C]]
```

```
class owlapy.owl_axiom.OwLNaryClassAxiom (
    class_expressions: List[owlapy.class_expression.OwLClassExpression],
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: OwLClassAxiom, OwLNaryAxiom[owlapy.class\_expression.OwLClassExpression]

Represents an axiom that contains two or more operands that could also be represented with multiple pairwise axioms.

```
__slots__ = '_class_expressions'
```

```
class_expressions () → Iterable[owlapy.class_expression.OwLClassExpression]
```

Gets all of the top level class expressions that appear in this axiom.

#### Returns

Sorted stream of class expressions that appear in the axiom.

```
as_pairwise_axioms () → Iterable[OwLNaryClassAxiom]
```

Gets this axiom as a set of pairwise axioms; if the axiom contains only two operands, the axiom itself is returned unchanged, including its annotations.

#### Returns

This axiom as a set of pairwise axioms.

```
__eq__ (other)
```

```
__hash__ ()
```

```
__repr__ ()
```

```
class owlapy.owl_axiom.OwLEquivalentClassesAxiom (
    class_expressions: List[owlapy.class_expression.OwLClassExpression],
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: OwLNaryClassAxiom

An equivalent classes axiom EquivalentClasses( CE<sub>1</sub> ... CE<sub>n</sub> ) states that all of the class expressions CE<sub>i</sub>, 1 ≤ i ≤ n, are semantically equivalent to each other. This axiom allows one to use each CE<sub>i</sub> as a synonym for each CE<sub>j</sub> — that is, in any expression in the ontology containing such an axiom, CE<sub>i</sub> can be replaced with CE<sub>j</sub> without affecting the meaning of the ontology.

([https://www.w3.org/TR/owl2-syntax/#Equivalent\\_Classes](https://www.w3.org/TR/owl2-syntax/#Equivalent_Classes))

```
__slots__ = ()
```

```
__iter__ ()
```

```
contains_named_equivalent_class () → bool
```



```

contains_owl_nothing() → bool

contains_owl_thing() → bool

named_classes() → Iterable[owlapy.class_expression.OWLClass]

```

**class** owlapy.owl\_axiom.OWLDisjointClassesAxiom(  
*class\_expressions: List[owlapy.class\_expression.OWLClassExpression],*  
*annotations: Iterable[OWLAnnotation] | None = None*)

Bases: *OWLNaryClassAxiom*

A disjoint classes axiom DisjointClasses( CE1 ... CEn ) states that all of the class expressions CE<sub>i</sub>, 1 ≤ i ≤ n, are pairwise disjoint; that is, no individual can be at the same time an instance of both CE<sub>i</sub> and CE<sub>j</sub> for i ≠ j.

([https://www.w3.org/TR/owl2-syntax/#Disjoint\\_Classes](https://www.w3.org/TR/owl2-syntax/#Disjoint_Classes))

```

__slots__ = ()

```

**class** owlapy.owl\_axiom.OWLNaryIndividualAxiom(  
*individuals: List[owlapy.owl\_individual.OWLIndividual],*  
*annotations: Iterable[OWLAnnotation] | None = None*)

Bases: *OWLIndividualAxiom, OWLNaryAxiom[owlapy.owl\_individual.OWLIndividual]*

Represents an axiom that contains two or more operands that could also be represented with multiple pairwise individual axioms.

```

__slots__ = '_individuals'

```

**individuals()** → Iterable[owlapy.owl\_individual.OWLIndividual]

Get the individuals.

**Returns**

Generator containing the individuals.

**as\_pairwise\_axioms()** → Iterable[OWLNaryIndividualAxiom]

```

__eq__(other)

__hash__()

__repr__()

```

**class** owlapy.owl\_axiom.OWLDifferentIndividualsAxiom(  
*individuals: List[owlapy.owl\_individual.OWLIndividual],*  
*annotations: Iterable[OWLAnnotation] | None = None*)

Bases: *OWLNaryIndividualAxiom*

An individual inequality axiom DifferentIndividuals( a1 ... an ) states that all of the individuals a<sub>i</sub>, 1 ≤ i ≤ n, are different from each other; that is, no individuals a<sub>i</sub> and a<sub>j</sub> with i ≠ j can be derived to be equal. This axiom can be used to axiomatize the unique name assumption — the assumption that all different individual names denote different individuals. ([https://www.w3.org/TR/owl2-syntax/#Individual\\_Inequality](https://www.w3.org/TR/owl2-syntax/#Individual_Inequality))

```

__slots__ = ()

```

**class** owlapy.owl\_axiom.OWLSameIndividualAxiom(  
*individuals: List[owlapy.owl\_individual.OWLIndividual],*  
*annotations: Iterable[OWLAnnotation] | None = None*)

Bases: *OWLNaryIndividualAxiom*

An individual equality axiom `SameIndividual( a1 ... an )` states that all of the individuals  $a_i$ ,  $1 \leq i \leq n$ , are equal to each other. This axiom allows one to use each  $a_i$  as a synonym for each  $a_j$  — that is, in any expression in the ontology containing such an axiom,  $a_i$  can be replaced with  $a_j$  without affecting the meaning of the ontology.

([https://www.w3.org/TR/owl2-syntax/#Individual\\_Equality](https://www.w3.org/TR/owl2-syntax/#Individual_Equality))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLNaryPropertyAxiom(properties: List[_P],
      annotations: Iterable[OWLAnnotation] | None = None)
```

```
Bases: Generic[_P], OWLPropertyAxiom, OWLNaryAxiom[_P]
```

Represents an axiom that contains two or more operands that could also be represented with multiple pairwise property axioms.

```
__slots__ = '_properties'
```

```
properties() → Iterable[_P]
```

Get all the properties that appear in the axiom.

#### Returns

Generator containing the properties.

```
as_pairwise_axioms() → Iterable[OWLNaryPropertyAxiom]
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLEquivalentObjectPropertiesAxiom(
      properties: List[owlapy.owl_property.OWLObjectPropertyExpression],
      annotations: Iterable[OWLAnnotation] | None = None)
```

```
Bases: OWLNaryPropertyAxiom[owlapy.owl_property.OWLObjectPropertyExpression], OWLObjectPropertyAxiom
```

An equivalent object properties axiom `EquivalentObjectProperties( OPE1 ... OPEn )` states that all of the object property expressions  $OPE_i$ ,  $1 \leq i \leq n$ , are semantically equivalent to each other. This axiom allows one to use each  $OPE_i$  as a synonym for each  $OPE_j$  — that is, in any expression in the ontology containing such an axiom,  $OPE_i$  can be replaced with  $OPE_j$  without affecting the meaning of the ontology.

([https://www.w3.org/TR/owl2-syntax/#Equivalent\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Equivalent_Object_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLDisjointObjectPropertiesAxiom(
      properties: List[owlapy.owl_property.OWLObjectPropertyExpression],
      annotations: Iterable[OWLAnnotation] | None = None)
```

```
Bases: OWLNaryPropertyAxiom[owlapy.owl_property.OWLObjectPropertyExpression], OWLObjectPropertyAxiom
```

A disjoint object properties axiom `DisjointObjectProperties( OPE1 ... OPEn )` states that all of the object property expressions  $OPE_i$ ,  $1 \leq i \leq n$ , are pairwise disjoint; that is, no individual  $x$  can be connected to an individual  $y$  by both  $OPE_i$  and  $OPE_j$  for  $i \neq j$ .

([https://www.w3.org/TR/owl2-syntax/#Disjoint\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Disjoint_Object_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLInverseObjectPropertiesAxiom(
    first: owlapy.owl_property.OWLObjectPropertyExpression,
    second: owlapy.owl_property.OWLObjectPropertyExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLNaryPropertyAxiom*[*owlapy.owl\_property.OWLObjectPropertyExpression*], *OWLObjectPropertyAxiom*

An inverse object properties axiom *InverseObjectProperties*( OPE1 OPE2 ) states that the object property expression OPE1 is an inverse of the object property expression OPE2. Thus, if an individual x is connected by OPE1 to an individual y, then y is also connected by OPE2 to x, and vice versa.

([https://www.w3.org/TR/owl2-syntax/#Inverse\\_Object\\_Properties\\_2](https://www.w3.org/TR/owl2-syntax/#Inverse_Object_Properties_2))

```
__slots__ = ('_first', '_second')
```

```
get_first_property() → owlapy.owl_property.OWLObjectPropertyExpression
```

```
get_second_property() → owlapy.owl_property.OWLObjectPropertyExpression
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLEquivalentDataPropertiesAxiom(
    properties: List[owlapy.owl_property.OWLDataPropertyExpression],
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLNaryPropertyAxiom*[*owlapy.owl\_property.OWLDataPropertyExpression*], *OWLDataPropertyAxiom*

An equivalent data properties axiom *EquivalentDataProperties*( DPE1 ... DPE<sub>n</sub> ) states that all the data property expressions DPE<sub>i</sub>, 1 ≤ i ≤ n, are semantically equivalent to each other. This axiom allows one to use each DPE<sub>i</sub> as a synonym for each DPE<sub>j</sub> — that is, in any expression in the ontology containing such an axiom, DPE<sub>i</sub> can be replaced with DPE<sub>j</sub> without affecting the meaning of the ontology.

([https://www.w3.org/TR/owl2-syntax/#Equivalent\\_Data\\_Properties](https://www.w3.org/TR/owl2-syntax/#Equivalent_Data_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLDisjointDataPropertiesAxiom(
    properties: List[owlapy.owl_property.OWLDataPropertyExpression],
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLNaryPropertyAxiom*[*owlapy.owl\_property.OWLDataPropertyExpression*], *OWLDataPropertyAxiom*

A disjoint data properties axiom *DisjointDataProperties*( DPE1 ... DPE<sub>n</sub> ) states that all of the data property expressions DPE<sub>i</sub>, 1 ≤ i ≤ n, are pairwise disjoint; that is, no individual x can be connected to a literal y by both

DPE<sub>i</sub> and DPE<sub>j</sub> for i ≠ j.

([https://www.w3.org/TR/owl2-syntax/#Disjoint\\_Data\\_Properties](https://www.w3.org/TR/owl2-syntax/#Disjoint_Data_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLSubClassOfAxiom(
    sub_class: owlapy.class_expression.OWLClassExpression,
    super_class: owlapy.class_expression.OWLClassExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLClassAxiom*

A subclass axiom *SubClassOf*( CE1 CE2 ) states that the class expression CE1 is a subclass of the class expression CE2. Roughly speaking, this states that CE1 is more specific than CE2. Subclass axioms are a fundamental type of axioms in OWL 2 and can be used to construct a class hierarchy. Other kinds of class expression axiom can be seen as syntactic shortcuts for one or more subclass axioms.

([https://www.w3.org/TR/owl2-syntax/#Subclass\\_Axioms](https://www.w3.org/TR/owl2-syntax/#Subclass_Axioms))

```
__slots__ = ('_sub_class', '_super_class')

property sub_class: owlapy.class_expression.OWLClassExpression

property super_class: owlapy.class_expression.OWLClassExpression

get_sub_class() → owlapy.class_expression.OWLClassExpression

get_super_class() → owlapy.class_expression.OWLClassExpression

__eq__(other)

__hash__()

__repr__()
```

```
class owlapy.owl_axiom.OWLDisjointUnionAxiom(cls_: owlapy.class_expression.OWLClass,
        class_expressions: List[owlapy.class_expression.OWLClassExpression],
        annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLClassAxiom*

A disjoint union axiom `DisjointUnion( C CE1 ... CEn )` states that a class `C` is a disjoint union of the class expressions `CEi`,  $1 \leq i \leq n$ , all of which are pairwise disjoint. Such axioms are sometimes referred to as covering axioms, as they state that the extensions of all `CEi` exactly cover the extension of `C`. Thus, each instance of `C` is an instance of exactly one `CEi`, and each instance of `CEi` is an instance of `C`.

([https://www.w3.org/TR/owl2-syntax/#Disjoint\\_Union\\_of\\_Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Disjoint_Union_of_Class_Expressions))

```
__slots__ = ('_cls', '_class_expressions')

get_owl_class() → owlapy.class_expression.OWLClass

get_class_expressions() → Iterable[owlapy.class_expression.OWLClassExpression]

get_owl_equivalent_classes_axiom() → OWLEquivalentClassesAxiom

get_owl_disjoint_classes_axiom() → OWLDisjointClassesAxiom

__eq__(other)

__hash__()

__repr__()
```

```
class owlapy.owl_axiom.OWLClassAssertionAxiom(
    individual: owlapy.owl_individual.OWLIndividual,
    class_expression: owlapy.class_expression.OWLClassExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLIndividualAxiom*

A class assertion `ClassAssertion( CE a )` states that the individual `a` is an instance of the class expression `CE`.

([https://www.w3.org/TR/owl2-syntax/#Class\\_Assertions](https://www.w3.org/TR/owl2-syntax/#Class_Assertions))

```
__slots__ = ('_individual', '_class_expression')

get_individual() → owlapy.owl_individual.OWLIndividual

get_class_expression() → owlapy.class_expression.OWLClassExpression
```

```

__eq__(other)

__hash__()

__repr__()

class owlapy.owl_axiom.OWLAnnotationProperty(iri: owlapy.iri.IRI | str)
    Bases: owlapy.owl_property.OWLProperty
    Represents an AnnotationProperty in the OWL 2 specification.

    __slots__ = '_iri'

    property iri: owlapy.iri.IRI
        Gets the IRI of this object.

        Returns
            The IRI of this object.

    property str: str
        Gets the string representation of this object

        Returns
            The IRI as string

class owlapy.owl_axiom.OWLAnnotation(property: OWLAnnotationProperty,
    value: owlapy.owl_annotation.OWLAnnotationValue)
    Bases: owlapy.owl_object.OWLObject
    Annotations are used in the various types of annotation axioms, which bind annotations to their subjects (i.e. axioms
    or declarations).

    __slots__ = ('_property', '_value')

    get_property() → OWLAnnotationProperty
        Gets the property that this annotation acts along.

        Returns
            The annotation property.

    get_value() → owlapy.owl_annotation.OWLAnnotationValue
        Gets the annotation value. The type of value will depend upon the type of the annotation e.g. whether the
        annotation is an OWLLiteral, an IRI or an OWLAnonymousIndividual.

        Returns
            The annotation value.

    __eq__(other)

    __hash__()

    __repr__()

class owlapy.owl_axiom.OWLAnnotationAxiom(annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLAxiom
    A super interface for annotation axioms.

    __slots__ = ()

    is_annotation_axiom() → bool

```

```
class owlapy.owl_axiom.OWLAnnotationAssertionAxiom(
    subject: owlapy.owl_annotation.OWLAnnotationSubject, annotation: OWLAnnotation,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLAnnotationAxiom*

An annotation assertion *AnnotationAssertion( AP as av )* states that the annotation subject as — an IRI or an anonymous individual — is annotated with the annotation property AP and the annotation value av.

([https://www.w3.org/TR/owl2-syntax/#Annotation\\_Assertion](https://www.w3.org/TR/owl2-syntax/#Annotation_Assertion))

```
__slots__ = ('_subject', '_annotation')
```

```
get_subject() → owlapy.owl_annotation.OWLAnnotationSubject
```

Gets the subject of this object.

**Returns**

The subject.

```
get_property() → OWLAnnotationProperty
```

Gets the property.

**Returns**

The property.

```
get_value() → owlapy.owl_annotation.OWLAnnotationValue
```

Gets the annotation value. This is either an IRI, an *OWLAnonymousIndividual* or an *OWLLiteral*.

**Returns**

The annotation value.

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLSubAnnotationPropertyOfAxiom(
    sub_property: OWLAnnotationProperty, super_property: OWLAnnotationProperty,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLAnnotationAxiom*

An annotation subproperty axiom *SubAnnotationPropertyOf( AP1 AP2 )* states that the annotation property AP1 is a subproperty of the annotation property AP2.

([https://www.w3.org/TR/owl2-syntax/#Annotation\\_Subproperties](https://www.w3.org/TR/owl2-syntax/#Annotation_Subproperties))

```
__slots__ = ('_sub_property', '_super_property')
```

```
get_sub_property() → OWLAnnotationProperty
```

```
get_super_property() → OWLAnnotationProperty
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLAnnotationPropertyDomainAxiom (property_: OWLAnnotationProperty,
    domain: owlapy.iri.IRI, annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLAnnotationAxiom*

An annotation property domain axiom AnnotationPropertyDomain( AP U ) states that the domain of the annotation property AP is the IRI U.

([https://www.w3.org/TR/owl2-syntax/#Annotation\\_Property\\_Domain](https://www.w3.org/TR/owl2-syntax/#Annotation_Property_Domain))

```
__slots__ = ('_property', '_domain')
```

```
get_property() → OWLAnnotationProperty
```

```
get_domain() → owlapy.iri.IRI
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLAnnotationPropertyRangeAxiom (property_: OWLAnnotationProperty,
    range_: owlapy.iri.IRI, annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *OWLAnnotationAxiom*

An annotation property range axiom AnnotationPropertyRange( AP U ) states that the range of the annotation property AP is the IRI U.

([https://www.w3.org/TR/owl2-syntax/#Annotation\\_Property\\_Range](https://www.w3.org/TR/owl2-syntax/#Annotation_Property_Range))

```
__slots__ = ('_property', '_range')
```

```
get_property() → OWLAnnotationProperty
```

```
get_range() → owlapy.iri.IRI
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLSubPropertyAxiom (sub_property: _P, super_property: _P,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: *Generic[\_P]*, *OWLPropertyAxiom*

Base interface for object and data sub-property axioms.

```
__slots__ = ('_sub_property', '_super_property')
```

```
get_sub_property() → _P
```

```
get_super_property() → _P
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLSubObjectPropertyOfAxiom(
    sub_property: owlapy.owl_property.OWLObjectPropertyExpression,
    super_property: owlapy.owl_property.OWLObjectPropertyExpression,
    annotations: Iterable[OWLAnnotation] | None = None)

Bases: OWLSubPropertyAxiom[owlapy.owl_property.OWLObjectPropertyExpression], OWLObjectPropertyAxiom
```

Object subproperty axioms are analogous to subclass axioms, and they come in two forms. The basic form is SubObjectPropertyOf( OPE1 OPE2 ). This axiom states that the object property expression OPE1 is a subproperty of the object property expression OPE2 — that is, if an individual *x* is connected by OPE1 to an individual *y*, then *x* is also connected by OPE2 to *y*. The more complex form is SubObjectPropertyOf( ObjectPropertyChain( OPE1 ... OPE<sub>n</sub> ) OPE ) but ObjectPropertyChain is not represented in owlapy yet.

([https://www.w3.org/TR/owl2-syntax/#Object\\_Subproperties](https://www.w3.org/TR/owl2-syntax/#Object_Subproperties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLSubDataPropertyOfAxiom(
    sub_property: owlapy.owl_property.OWLDataPropertyExpression,
    super_property: owlapy.owl_property.OWLDataPropertyExpression,
    annotations: Iterable[OWLAnnotation] | None = None)

Bases: OWLSubPropertyAxiom[owlapy.owl_property.OWLDataPropertyExpression], OWLDataPropertyAxiom
```

A data subproperty axiom SubDataPropertyOf( DPE1 DPE2 ) states that the data property expression DPE1 is a subproperty of the data property expression DPE2 — that is, if an individual *x* is connected by DPE1 to a literal *y*, then *x* is connected by DPE2 to *y* as well.

([https://www.w3.org/TR/owl2-syntax/#Data\\_Subproperties](https://www.w3.org/TR/owl2-syntax/#Data_Subproperties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLPropertyAssertionAxiom(
    subject: owlapy.owl_individual.OWLIndividual, property_: _P, object_: _C,
    annotations: Iterable[OWLAnnotation] | None = None)

Bases: Generic[_P, _C], OWLIndividualAxiom
```

Base class for Property Assertion axioms.

```
__slots__ = ('_subject', '_property', '_object')
```

```
get_subject() → owlapy.owl_individual.OWLIndividual
```

```
get_property() → _P
```

```
get_object() → _C
```

```
__eq__(other)
```

```
__hash__()
```

```
__repr__()
```

```
class owlapy.owl_axiom.OWLObjectPropertyAssertionAxiom(
    subject: owlapy.owl_individual.OWLIndividual,
    property_: owlapy.owl_property.OWLObjectPropertyExpression,
    object_: owlapy.owl_individual.OWLIndividual,
    annotations: Iterable[OWLAnnotation] | None = None)
```



Bases: `OWLPropertyAssertionAxiom[owlapy.owl_property.OWLObjectPropertyExpression, owlapy.owl_individual.OWLIndividual]`

A positive object property assertion `ObjectPropertyAssertion( OPE a1 a2 )` states that the individual a1 is connected by the object property expression OPE to the individual a2.

([https://www.w3.org/TR/owl2-syntax/#Positive\\_Object\\_Property\\_Assertions](https://www.w3.org/TR/owl2-syntax/#Positive_Object_Property_Assertions))

`__slots__ = ()`

```
class owlapy.owl_axiom.OWLNegativeObjectPropertyAssertionAxiom(  
    subject: owlapy.owl_individual.OWLIndividual,  
    property_: owlapy.owl_property.OWLObjectPropertyExpression,  
    object_: owlapy.owl_individual.OWLIndividual,  
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyAssertionAxiom[owlapy.owl_property.OWLObjectPropertyExpression, owlapy.owl_individual.OWLIndividual]`

A negative object property assertion `NegativeObjectPropertyAssertion( OPE a1 a2 )` states that the individual a1 is not connected by the object property expression OPE to the individual a2.

([https://www.w3.org/TR/owl2-syntax/#Negative\\_Object\\_Property\\_Assertions](https://www.w3.org/TR/owl2-syntax/#Negative_Object_Property_Assertions))

`__slots__ = ()`

```
class owlapy.owl_axiom.OWLDataPropertyAssertionAxiom(  
    subject: owlapy.owl_individual.OWLIndividual,  
    property_: owlapy.owl_property.OWLDataPropertyExpression,  
    object_: owlapy.owl_literal.OWLLiteral, annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyAssertionAxiom[owlapy.owl_property.OWLDataPropertyExpression, owlapy.owl_literal.OWLLiteral]`

A positive data property assertion `DataPropertyAssertion( DPE a lt )` states that the individual a is connected by the data property expression DPE to the literal lt.

([https://www.w3.org/TR/owl2-syntax/#Positive\\_Data\\_Property\\_Assertions](https://www.w3.org/TR/owl2-syntax/#Positive_Data_Property_Assertions))

`__slots__ = ()`

```
class owlapy.owl_axiom.OWLNegativeDataPropertyAssertionAxiom(  
    subject: owlapy.owl_individual.OWLIndividual,  
    property_: owlapy.owl_property.OWLDataPropertyExpression,  
    object_: owlapy.owl_literal.OWLLiteral, annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyAssertionAxiom[owlapy.owl_property.OWLDataPropertyExpression, owlapy.owl_literal.OWLLiteral]`

A negative data property assertion `NegativeDataPropertyAssertion( DPE a lt )` states that the individual a is not connected by the data property expression DPE to the literal lt.

([https://www.w3.org/TR/owl2-syntax/#Negative\\_Data\\_Property\\_Assertions](https://www.w3.org/TR/owl2-syntax/#Negative_Data_Property_Assertions))

`__slots__ = ()`

```
class owlapy.owl_axiom.OWLUnaryPropertyAxiom(property_: _P,  
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `Generic[_P], OWLPropertyAxiom`

Base class for Unary property axiom.

```

__slots__ = '_property'

get_property() → _P

class owlapy.owl_axiom.OwlObjectPropertyCharacteristicAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)

Bases: OwlUnaryPropertyAxiom[owlapy.owl_property.OwlObjectPropertyExpression], OwlOb-
jectPropertyAxiom

Base interface for functional object property axiom.

__slots__ = ()

__eq__(other)

__hash__()

__repr__()

class owlapy.owl_axiom.OwlFunctionalObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)

Bases: OwlObjectPropertyCharacteristicAxiom

An object property functionality axiom FunctionalObjectProperty( OPE ) states that the object property expression
OPE is functional — that is, for each individual x, there can be at most one distinct individual y such that x is
connected by OPE to y.

(https://www.w3.org/TR/owl2-syntax/#Functional\_Object\_Properties)

__slots__ = ()

class owlapy.owl_axiom.OwlAsymmetricObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)

Bases: OwlObjectPropertyCharacteristicAxiom

An object property asymmetry axiom AsymmetricObjectProperty( OPE ) states that the object property expression
OPE is asymmetric — that is, if an individual x is connected by OPE to an individual y, then y cannot be connected
by OPE to x.

(https://www.w3.org/TR/owl2-syntax/#Symmetric\_Object\_Properties)

__slots__ = ()

class owlapy.owl_axiom.OwlInverseFunctionalObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)

Bases: OwlObjectPropertyCharacteristicAxiom

An object property inverse functionality axiom InverseFunctionalObjectProperty( OPE ) states that the object
property expression OPE is inverse-functional — that is, for each individual x, there can be at most one individual
y such that y is connected by OPE with x.

(https://www.w3.org/TR/owl2-syntax/#Inverse-Functional\_Object\_Properties)

__slots__ = ()

```

```
class owlapy.owl_axiom.OwlIrreflexiveObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)
```

Bases: *OwlObjectPropertyCharacteristicAxiom*

An object property irreflexivity axiom *IrreflexiveObjectProperty( OPE )* states that the object property expression OPE is irreflexive — that is, no individual is connected by OPE to itself.

([https://www.w3.org/TR/owl2-syntax/#Irreflexive\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Irreflexive_Object_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OwlReflexiveObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)
```

Bases: *OwlObjectPropertyCharacteristicAxiom*

An object property reflexivity axiom *ReflexiveObjectProperty( OPE )* states that the object property expression OPE is reflexive — that is, each individual is connected by OPE to itself. Each such axiom can be seen as a syntactic shortcut for the following axiom: *SubClassOf( owl:Thing ObjectHasSelf( OPE ) )*

([https://www.w3.org/TR/owl2-syntax/#Reflexive\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Reflexive_Object_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OwlSymmetricObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)
```

Bases: *OwlObjectPropertyCharacteristicAxiom*

An object property symmetry axiom *SymmetricObjectProperty( OPE )* states that the object property expression OPE is symmetric — that is, if an individual x is connected by OPE to an individual y, then y is also connected by OPE to x. Each such axiom can be seen as a syntactic shortcut for the following axiom:

*SubObjectPropertyOf( OPE ObjectInverseOf( OPE ) )*

([https://www.w3.org/TR/owl2-syntax/#Symmetric\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Symmetric_Object_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OwlTransitiveObjectPropertyAxiom(
    property_: owlapy.owl_property.OwlObjectPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)
```

Bases: *OwlObjectPropertyCharacteristicAxiom*

An object property transitivity axiom *TransitiveObjectProperty( OPE )* states that the object property expression OPE is transitive — that is, if an individual x is connected by OPE to an individual y that is connected by OPE to an individual z, then x is also connected by OPE to z. Each such axiom can be seen as a syntactic shortcut for the following axiom: *SubObjectPropertyOf( ObjectPropertyChain( OPE OPE ) OPE )*

([https://www.w3.org/TR/owl2-syntax/#Transitive\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Transitive_Object_Properties))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OwlDataPropertyCharacteristicAxiom(
    property_: owlapy.owl_property.OwlDataPropertyExpression,
    annotations: Iterable[OwlAnnotation] | None = None)
```

Bases: *OwlUnaryPropertyAxiom[owlapy.owl\_property.OwlDataPropertyExpression]*, *OwlDataPropertyAxiom*

Base interface for Functional data property axiom.

```

__slots__ = ()

__eq__(other)

__hash__()

__repr__()

class owlapy.owl_axiom.OWLFunctionalDataPropertyAxiom (
    property_: owlapy.owl_property.OWLDataPropertyExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
    Bases: OWLDataPropertyCharacteristicAxiom

    A data property functionality axiom FunctionalDataProperty( DPE ) states that the data property expression DPE
    is functional — that is, for each individual x, there can be at most one distinct literal y such that x is connected by
    DPE with y. Each such axiom can be seen as a syntactic shortcut for the following axiom: SubClassOf( owl:Thing
    DataMaxCardinality( 1 DPE ) )

    (https://www.w3.org/TR/owl2-syntax/#Transitive\_Object\_Properties)

__slots__ = ()

class owlapy.owl_axiom.OWLPropertyDomainAxiom (property_: _P,
    domain: owlapy.class_expression.OWLClassExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
    Bases: Generic[_P], OWLUnaryPropertyAxiom[_P]

    Base class for Property Domain axioms.

__slots__ = '_domain'

get_domain() → owlapy.class_expression.OWLClassExpression

__eq__(other)

__hash__()

__repr__()

class owlapy.owl_axiom.OWLPropertyRangeAxiom (property_: _P, range_: _R,
    annotations: Iterable[OWLAnnotation] | None = None)
    Bases: Generic[_P, _R], OWLUnaryPropertyAxiom[_P]

    Base class for Property Range axioms.

__slots__ = '_range'

property prop

property range

get_range() → _R

__eq__(other)

__hash__()

__repr__()

```

```
class owlapy.owl_axiom.OWLObjectPropertyDomainAxiom(
    property_: owlapy.owl_property.OWLObjectPropertyExpression,
    domain: owlapy.class_expression.OWLClassExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyDomainAxiom[owlapy.owl_property.OWLObjectPropertyExpression]`

An object property domain axiom `ObjectPropertyDomain( OPE CE )` states that the domain of the object property expression OPE is the class expression CE — that is, if an individual x is connected by OPE with some other individual, then x is an instance of CE. Each such axiom can be seen as a syntactic shortcut for the following axiom: `SubClassOf( ObjectSomeValuesFrom( OPE owl:Thing ) CE )`

([https://www.w3.org/TR/owl2-syntax/#Object\\_Property\\_Domain](https://www.w3.org/TR/owl2-syntax/#Object_Property_Domain))

```
__slots__ = ()
```

```
property prop
```

```
class owlapy.owl_axiom.OWLDataPropertyDomainAxiom(
    property_: owlapy.owl_property.OWLDataPropertyExpression,
    domain: owlapy.class_expression.OWLClassExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyDomainAxiom[owlapy.owl_property.OWLDataPropertyExpression]`

A data property domain axiom `DataPropertyDomain( DPE CE )` states that the domain of the data property expression DPE is the class expression CE — that is, if an individual x is connected by DPE with some literal, then x is an instance of CE. Each such axiom can be seen as a syntactic shortcut for the following axiom: `SubClassOf( DataSomeValuesFrom( DPE rdfs:Literal ) CE )`

([https://www.w3.org/TR/owl2-syntax/#Data\\_Property\\_Domain](https://www.w3.org/TR/owl2-syntax/#Data_Property_Domain))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLObjectPropertyRangeAxiom(
    property_: owlapy.owl_property.OWLObjectPropertyExpression,
    range_: owlapy.class_expression.OWLClassExpression,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyRangeAxiom[owlapy.owl_property.OWLObjectPropertyExpression, owlapy.class_expression.OWLClassExpression]`

An object property range axiom `ObjectPropertyRange( OPE CE )` states that the range of the object property expression OPE is the class expression CE — that is, if some individual is connected by OPE with an individual x, then x is an instance of CE. Each such axiom can be seen as a syntactic shortcut for the following axiom: `SubClassOf( owl:Thing ObjectAllValuesFrom( OPE CE ) )`

([https://www.w3.org/TR/owl2-syntax/#Object\\_Property\\_Range](https://www.w3.org/TR/owl2-syntax/#Object_Property_Range))

```
__slots__ = ()
```

```
class owlapy.owl_axiom.OWLDataPropertyRangeAxiom(
    property_: owlapy.owl_property.OWLDataPropertyExpression,
    range_: owlapy.owl_datatype.OWLDataRange,
    annotations: Iterable[OWLAnnotation] | None = None)
```

Bases: `OWLPropertyRangeAxiom[owlapy.owl_property.OWLDataPropertyExpression, owlapy.owl_datatype.OWLDataRange]`

A data property range axiom `DataPropertyRange( DPE DR )` states that the range of the data property expression DPE is the data range DR — that is, if some individual is connected by DPE with a literal x, then x is in DR. The arity of DR must be one. Each such axiom can be seen as a syntactic shortcut for the following axiom: `SubClassOf( owl:Thing DataAllValuesFrom( DPE DR ) )`

([https://www.w3.org/TR/owl2-syntax/#Data\\_Property\\_Range](https://www.w3.org/TR/owl2-syntax/#Data_Property_Range))

```
__slots__ = ()
```

## owlapy.owl\_data\_ranges

OWL Data Ranges

[https://www.w3.org/TR/owl2-syntax/#Data\\_Ranges](https://www.w3.org/TR/owl2-syntax/#Data_Ranges)

DataRange := Datatype | DataIntersectionOf | DataUnionOf | DataComplementOf | DataOneOf | DatatypeRestriction

## Classes

<i>OWLPropertyRange</i>	OWL Objects that can be the ranges of properties.
<i>OWLDataRange</i>	Represents a DataRange in the OWL 2 Specification.
<i>OWLNaryDataRange</i>	OWLNaryDataRange.
<i>OWLDataIntersectionOf</i>	An intersection data range DataIntersectionOf( DR1 ... DRn ) contains all tuples of literals that are contained
<i>OWLDataUnionOf</i>	A union data range DataUnionOf( DR1 ... DRn ) contains all tuples of literals that are contained in the at least
<i>OWLDataComplementOf</i>	A complement data range DataComplementOf( DR ) contains all tuples of literals that are not contained in the

## Module Contents

```
class owlapy.owl_data_ranges.OWLPropertyRange
```

Bases: *owlapy.owl\_object.OWLObject*

OWL Objects that can be the ranges of properties.

```
class owlapy.owl_data_ranges.OWLDataRange
```

Bases: *OWLPropertyRange*

Represents a DataRange in the OWL 2 Specification.

```
class owlapy.owl_data_ranges.OWLNaryDataRange(operands: Iterable[OWLDataRange])
```

Bases: *OWLDataRange*, *owlapy.meta\_classes.HasOperands[OWLDataRange]*

OWLNaryDataRange.

```
__slots__ = ()
```

```
operands() → Iterable[OWLDataRange]
```

Gets the operands - e.g., the individuals in a sameAs axiom, or the classes in an equivalent classes axiom.

### Returns

The operands.

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
class owlapy.owl_data_ranges.OWLDataIntersectionOf (operands: Iterable[OWLDataRange])
```

Bases: *OWLNaryDataRange*

An intersection data range *DataIntersectionOf*( DR1 ... DRn ) contains all tuples of literals that are contained in each data range DRi for  $1 \leq i \leq n$ . All data ranges DRi must be of the same arity, and the resulting data range is of that arity as well.

([https://www.w3.org/TR/owl2-syntax/#Intersection\\_of\\_Data\\_Ranges](https://www.w3.org/TR/owl2-syntax/#Intersection_of_Data_Ranges))

```
__slots__ = '_operands'
```

```
type_index: Final = 4004
```

```
class owlapy.owl_data_ranges.OWLDataUnionOf (operands: Iterable[OWLDataRange])
```

Bases: *OWLNaryDataRange*

A union data range *DataUnionOf*( DR1 ... DRn ) contains all tuples of literals that are contained in the at least one data range DRi for  $1 \leq i \leq n$ . All data ranges DRi must be of the same arity, and the resulting data range is of that arity as well.

([https://www.w3.org/TR/owl2-syntax/#Union\\_of\\_Data\\_Ranges](https://www.w3.org/TR/owl2-syntax/#Union_of_Data_Ranges))

```
__slots__ = '_operands'
```

```
type_index: Final = 4005
```

```
class owlapy.owl_data_ranges.OWLDataComplementOf (data_range: OWLDataRange)
```

Bases: *OWLDataRange*

A complement data range *DataComplementOf*( DR ) contains all tuples of literals that are not contained in the data range DR. The resulting data range has the arity equal to the arity of DR.

([https://www.w3.org/TR/owl2-syntax/#Complement\\_of\\_Data\\_Ranges](https://www.w3.org/TR/owl2-syntax/#Complement_of_Data_Ranges))

```
type_index: Final = 4002
```

```
get_data_range() → OWLDataRange
```

#### Returns

The wrapped data range.

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

## owlapy.owl\_datatype

OWL Datatype

### Classes

---

*OWLDatatype*

Datatypes are entities that refer to sets of data values. Thus, datatypes are analogous to classes,

---

## Module Contents

**class** owlapy.owl\_datatype.**OWLDatatype** (*iri: owlapy.iri.IRI | owlapy.meta\_classes.HasIRI*)

Bases: *owlapy.owl\_object.OWLEntity, owlapy.owl\_data\_ranges.OWLDataRange*

Datatypes are entities that refer to sets of data values. Thus, datatypes are analogous to classes, the main difference being that the former contain data values such as strings and numbers, rather than individuals. Datatypes are a kind of data range, which allows them to be used in restrictions. Each data range is associated with an arity; for datatypes, the arity is always one. The built-in datatype `rdfs:Literal` denotes any set of data values that contains the union of the value spaces of all datatypes.

(<https://www.w3.org/TR/owl2-syntax/#Datatypes>)

**\_\_slots\_\_** = `'_iri'`

**type\_index**: `Final = 4001`

**property iri**: *owlapy.iri.IRI*

Gets the IRI of this object.

### Returns

The IRI of this object.

**property str**: `str`

Gets the string representation of this object

### Returns

The IRI as string

## owlapy.owl\_hierarchy

Classes representing hierarchy in OWL.

## Classes

<i>AbstractHierarchy</i>	Representation of an abstract hierarchy which can be used for classes or properties.
<i>ClassHierarchy</i>	Representation of a class hierarchy.
<i>ObjectPropertyHierarchy</i>	Representation of an object property hierarchy.
<i>DatatypePropertyHierarchy</i>	Representation of a data property hierarchy.

## Module Contents

**class** owlapy.owl\_hierarchy.**AbstractHierarchy** (*factory: Type[\_S],  
hierarchy\_down: Iterable[Tuple[\_S, Iterable[\_S]]]*)

**class** owlapy.owl\_hierarchy.**AbstractHierarchy** (*factory: Type[\_S],  
reasoner: owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner*)

Bases: `Generic[_S]`

Representation of an abstract hierarchy which can be used for classes or properties.

### Parameters

- **hierarchy\_down** – A downwards hierarchy given as a mapping of Entities to sub-entities.
- **reasoner** – Alternatively, a reasoner whose `root_ontology` is queried for entities.



```
__slots__ = ('_Type', '_ent_set', '_parents_map', '_parents_map_trans',
'_children_map', ...)
```

```
classmethod get_top_entity() → _S
```

#### Abstractmethod

The most general entity in this hierarchy, which contains all the entities.

```
classmethod get_bottom_entity() → _S
```

#### Abstractmethod

The most specific entity in this hierarchy, which contains none of the entities.

```
static restrict (hierarchy: _U, *, remove: Iterable[_S] = None, allow: Iterable[_S] = None) → _U
```

Restrict a given hierarchy to a set of allowed/removed entities.

#### Parameters

- **hierarchy** – An existing Entity hierarchy to restrict.
- **remove** – Set of entities which should be ignored.
- **allow** – Set of entities which should be used.

#### Returns

The restricted hierarchy.

```
restrict_and_copy (*, remove: Iterable[_S] = None, allow: Iterable[_S] = None) → _U
```

Restrict this hierarchy.

See restrict for more info.

```
parents (entity: _S, direct: bool = True) → Iterable[_S]
```

Parents of an entity.

#### Parameters

- **entity** – Entity for which to query parent entities.
- **direct** – False to return transitive parents.

#### Returns

Super-entities.

```
is_parent_of (a: _S, b: _S) → bool
```

if A is a parent of B.

#### Note

A is always a parent of A.

```
is_child_of (a: _S, b: _S) → bool
```

If A is a child of B.

#### Note

A is always a child of A.

**children** (*entity*: *\_S*, *direct*: *bool* = *True*) → *Iterable[\_S]*

Children of an entity.

#### Parameters

- **entity** – Entity for which to query child entities.
- **direct** – False to return transitive children.

#### Returns

Sub-entities.

**siblings** (*entity*: *\_S*) → *Iterable[\_S]*

**items** () → *Iterable[\_S]*

**roots** (*of*: *\_S* | *None* = *None*) → *Iterable[\_S]*

**leaves** (*of*: *\_S* | *None* = *None*) → *Iterable[\_S]*

**\_\_contains\_\_** (*item*: *\_S*) → *bool*

**\_\_len\_\_** ()

```
class owlapy.owl_hierarchy.ClassHierarchy(  
    hierarchy_down: Iterable[Tuple[owlapy.class_expression.OWLClass, Iterable[owlapy.class_expression.OWLClass]]])
```

```
class owlapy.owl_hierarchy.ClassHierarchy(  
    reasoner: owlapy.abstracts.abstract_owl_reasoner.AbstractOWLReasoner)
```

Bases: *AbstractHierarchy*[*owlapy.class\_expression.OWLClass*]

Representation of a class hierarchy.

#### Parameters

- **hierarchy\_down** – A downwards hierarchy given as a mapping of Class to sub-classes.
- **reasoner** – Alternatively, a reasoner whose *root\_ontology* is queried for classes and sub-classes.

**classmethod get\_top\_entity** () → *owlapy.class\_expression.OWLClass*

The most general entity in this hierarchy, which contains all the entities.

**classmethod get\_bottom\_entity** () → *owlapy.class\_expression.OWLClass*

The most specific entity in this hierarchy, which contains none of the entities.

**sub\_classes** (*entity*: *owlapy.class\_expression.OWLClass*, *direct*: *bool* = *True*)  
→ *Iterable[owlapy.class\_expression.OWLClass]*

**super\_classes** (*entity*: *owlapy.class\_expression.OWLClass*, *direct*: *bool* = *True*)  
→ *Iterable[owlapy.class\_expression.OWLClass]*

**is\_subclass\_of** (*subclass*: *owlapy.class\_expression.OWLClass*,  
*superclass*: *owlapy.class\_expression.OWLClass*) → *bool*

```
class owlapy.owl_hierarchy.ObjectPropertyHierarchy(  
    hierarchy_down: Iterable[Tuple[owlapy.owl_property.OWLObjectProperty, Iterable[owlapy.owl_property.OWLObjectProperty]]])
```

```

class owlapy.owl_hierarchy.ObjectPropertyHierarchy (
    reasoner: owlapy.abstracts.abstract_owl_reasoner.AbstractOWLReasoner)
Bases: AbstractHierarchy[owlapy.owl_property.OWLObjectProperty]
Representation of an objet property hierarchy.

classmethod get_top_entity () → owlapy.owl_property.OWLObjectProperty
    The most general entity in this hierarchy, which contains all the entities.

classmethod get_bottom_entity () → owlapy.owl_property.OWLObjectProperty
    The most specific entity in this hierarchy, which contains none of the entities.

sub_object_properties (entity: owlapy.owl_property.OWLObjectProperty, direct: bool = True)
    → Iterable[owlapy.owl_property.OWLObjectProperty]

super_object_properties (entity: owlapy.owl_property.OWLObjectProperty, direct: bool = True)
    → Iterable[owlapy.owl_property.OWLObjectProperty]

more_general_roles (role: owlapy.owl_property.OWLObjectProperty, direct: bool = True)
    → Iterable[owlapy.owl_property.OWLObjectProperty]

more_special_roles (role: owlapy.owl_property.OWLObjectProperty, direct: bool = True)
    → Iterable[owlapy.owl_property.OWLObjectProperty]

is_sub_property_of (sub_property: owlapy.owl_property.OWLObjectProperty,
    super_property: owlapy.owl_property.OWLObjectProperty) → bool

most_general_roles () → Iterable[owlapy.owl_property.OWLObjectProperty]

most_special_roles () → Iterable[owlapy.owl_property.OWLObjectProperty]

class owlapy.owl_hierarchy.DatatypePropertyHierarchy (
    hierarchy_down: Iterable[Tuple[owlapy.owl_property.OWLDataProperty, Iterable[owlapy.owl_property.OWLDataProperty]]])

class owlapy.owl_hierarchy.DatatypePropertyHierarchy (
    reasoner: owlapy.abstracts.abstract_owl_reasoner.AbstractOWLReasoner)
Bases: AbstractHierarchy[owlapy.owl_property.OWLDataProperty]
Representation of a data property hierarchy.

classmethod get_top_entity () → owlapy.owl_property.OWLDataProperty
    The most general entity in this hierarchy, which contains all the entities.

classmethod get_bottom_entity () → owlapy.owl_property.OWLDataProperty
    The most specific entity in this hierarchy, which contains none of the entities.

sub_data_properties (entity: owlapy.owl_property.OWLDataProperty, direct: bool = True)

super_data_properties (entity: owlapy.owl_property.OWLDataProperty, direct: bool = True)

more_general_roles (role: owlapy.owl_property.OWLDataProperty, direct: bool = True)
    → Iterable[owlapy.owl_property.OWLDataProperty]

more_special_roles (role: owlapy.owl_property.OWLDataProperty, direct: bool = True)
    → Iterable[owlapy.owl_property.OWLDataProperty]

is_sub_property_of (sub_property: owlapy.owl_property.OWLDataProperty,
    super_property: owlapy.owl_property.OWLDataProperty) → bool

```

`most_general_roles()` → Iterable[*owlapy.owl\_property.OWLDataProperty*]

`most_special_roles()` → Iterable[*owlapy.owl\_property.OWLDataProperty*]

## owlapy.owl\_individual

OWL Individuals

### Classes

<i>OWLIndividual</i>	Represents a named or anonymous individual.
<i>OWLNamedIndividual</i>	Named individuals are identified using an IRI. Since they are given an IRI, named individuals are entities.

### Module Contents

**class** owlapy.owl\_individual.OWLIndividual

Bases: *owlapy.owl\_object.OWLObject*

Represents a named or anonymous individual.

`__slots__ = ()`

**class** owlapy.owl\_individual.OWLNamedIndividual (*iri: owlapy.iri.IRI | str*)

Bases: *OWLIndividual*, *owlapy.owl\_object.OWLEntity*

Named individuals are identified using an IRI. Since they are given an IRI, named individuals are entities. IRIs from the reserved vocabulary must not be used to identify named individuals in an OWL 2 DL ontology.

([https://www.w3.org/TR/owl2-syntax/#Named\\_Individuals](https://www.w3.org/TR/owl2-syntax/#Named_Individuals))

`__slots__ = '_iri'`

`type_index: Final = 1005`

**property** *iri: owlapy.iri.IRI*

Gets the IRI of this object.

#### Returns

The IRI of this object.

**property** *str*

Gets the string representation of this object

#### Returns

The IRI as string

**property** *reminder*

## owlapy.owl\_literal

OWL Literals



## Attributes

<i>OWLTopObjectProperty</i>
<i>OWLBottomObjectProperty</i>
<i>OWLTopDataProperty</i>
<i>OWLBottomDataProperty</i>
<i>DoubleOWLDatatype</i>
<i>FloatOWLDatatype</i>
<i>DecimalOWLDatatype</i>
<i>IntegerOWLDatatype</i>
<i>NonNegativeIntegerOWLDatatype</i>
<i>NonPositiveIntegerOWLDatatype</i>
<i>NegativeIntegerOWLDatatype</i>
<i>PositiveIntegerOWLDatatype</i>
<i>BooleanOWLDatatype</i>
<i>StringOWLDatatype</i>
<i>DateOWLDatatype</i>
<i>TimeOWLDatatype</i>
<i>GYearMonthOWLDatatype</i>
<i>GMonthDayOWLDatatype</i>
<i>GYearOWLDatatype</i>
<i>GMonthOWLDatatype</i>
<i>GDayOWLDatatype</i>
<i>DateTimeOWLDatatype</i>
<i>DurationOWLDatatype</i>
<i>TopOWLDatatype</i>
<i>NUMERIC_DATATYPES</i>
<i>TIME_DATATYPES</i>
<i>Literals</i>

## Classes

<i>FloatSpecialValue</i>	Generic enumeration.
<i>OWLLiteral</i>	Literals represent data values such as particular strings or integers. They are analogous to typed RDF

## Module Contents

```
owlapy.owl_literal.OWLTopObjectProperty: Final
owlapy.owl_literal.OWLBottomObjectProperty: Final
owlapy.owl_literal.OWLTopDataProperty: Final
owlapy.owl_literal.OWLBottomDataProperty: Final
owlapy.owl_literal.DoubleOWLDatatype: Final
owlapy.owl_literal.FloatOWLDatatype: Final
owlapy.owl_literal.DecimalOWLDatatype: Final
owlapy.owl_literal.IntegerOWLDatatype: Final
owlapy.owl_literal.NonNegativeIntegerOWLDatatype: Final
owlapy.owl_literal.NonPositiveIntegerOWLDatatype: Final
owlapy.owl_literal.NegativeIntegerOWLDatatype: Final
owlapy.owl_literal.PositiveIntegerOWLDatatype: Final
owlapy.owl_literal.BooleanOWLDatatype: Final
owlapy.owl_literal.StringOWLDatatype: Final
owlapy.owl_literal.DateOWLDatatype: Final
owlapy.owl_literal.TimeOWLDatatype: Final
owlapy.owl_literal.GYearMonthOWLDatatype: Final
owlapy.owl_literal.GMonthDayOWLDatatype: Final
owlapy.owl_literal.GYearOWLDatatype: Final
owlapy.owl_literal.GMonthOWLDatatype: Final
owlapy.owl_literal.GDayOWLDatatype: Final
owlapy.owl_literal.DateTimeOWLDatatype: Final
owlapy.owl_literal.DurationOWLDatatype: Final
owlapy.owl_literal.TopOWLDatatype: Final
owlapy.owl_literal.NUMERIC_DATATYPES: Final[Set[owlapy.owl_datatype.OWLDatatype]]
```

```
owlapy.owl_literal.TIME_DATATYPES: Final[Set[owlapy.owl_datatype.OWLDatatype]]
```

```
class owlapy.owl_literal.FloatSpecialValue
```

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

```
NAN = 'Nan'
```

```
POS_INF = 'INF'
```

```
NEG_INF = '-INF'
```

```
__str__()
```

```
owlapy.owl_literal.Literals
```

```
class owlapy.owl_literal.OWLLiteral
```

Bases: `owlapy.owl_annotation.OWLAnnotationValue`

Literals represent data values such as particular strings or integers. They are analogous to typed RDF literals and can also be understood as individuals denoting data values. Each literal consists of a lexical form, which is a string, and a datatype.

(<https://www.w3.org/TR/owl2-syntax/#Literals>)

```
__slots__ = ()
```

```
type_index: Final = 4008
```

```
get_literal() → str
```

Gets the lexical value of this literal. Note that the language tag is not included.

#### Returns

The lexical value of this literal.

```
is_boolean() → bool
```

Whether this literal is typed as boolean.

```
parse_boolean() → bool
```

Parses the lexical value of this literal into a bool. The lexical value of this literal should be in the lexical space of the boolean datatype ("<http://www.w3.org/2001/XMLSchema#boolean>").

#### Returns

A bool value that is represented by this literal.

```
is_double() → bool
```

Whether this literal is typed as double.

```
parse_double() → float
```

Parses the lexical value of this literal into a double. The lexical value of this literal should be in the lexical space of the double datatype ("[https://www.w3.org/TR/owl2-syntax/#Floating-Point\\_Numbers](https://www.w3.org/TR/owl2-syntax/#Floating-Point_Numbers)").

#### Returns

A double value that is represented by this literal.

```
is_float() → bool
```

Whether this literal is typed as float.



**parse\_float()** → float

Parses the lexical value of this literal into a float. The lexical value of this literal should be in the lexical space of the float datatype ("[https://www.w3.org/TR/owl2-syntax/#Floating-Point\\_Numbers](https://www.w3.org/TR/owl2-syntax/#Floating-Point_Numbers)").

**Returns**

A float value that is represented by this literal.

**is\_decimal()** → bool

Whether this literal is typed as decimal.

**parse\_decimal()** → decimal.Decimal

Parses the lexical value of this literal into a decimal. The lexical value of this literal should be in the lexical space of the decimal datatype ("[https://www.w3.org/TR/owl2-syntax/#Floating-Point\\_Numbers](https://www.w3.org/TR/owl2-syntax/#Floating-Point_Numbers)").

**Returns**

A decimal value that is represented by this literal.

**is\_integer()** → bool

Whether this literal is typed as integer.

**parse\_integer()** → int

Parses the lexical value of this literal into an integer. The lexical value of this literal should be in the lexical space of the integer datatype ("<http://www.w3.org/2001/XMLSchema#integer>").

**Returns**

An integer value that is represented by this literal.

**is\_string()** → bool

Whether this literal is typed as string.

**parse\_string()** → str

Parses the lexical value of this literal into a string. The lexical value of this literal should be in the lexical space of the string datatype ("<http://www.w3.org/2001/XMLSchema#string>").

**Returns**

A string value that is represented by this literal.

**is\_date()** → bool

Whether this literal is typed as date.

**parse\_date()** → datetime.date

Parses the lexical value of this literal into a date. The lexical value of this literal should be in the lexical space of the date datatype ("<http://www.w3.org/2001/XMLSchema#date>").

**Returns**

A date value that is represented by this literal.

**is\_datetime()** → bool

Whether this literal is typed as dateTime.

**parse\_datetime()** → datetime.datetime

Parses the lexical value of this literal into a datetime. The lexical value of this literal should be in the lexical space of the dateTime datatype ("<http://www.w3.org/2001/XMLSchema#dateTime>").

**Returns**

A datetime value that is represented by this literal.

**is\_duration()** → bool

Whether this literal is typed as duration.

**parse\_duration()** → pandas.Timedelta

Parses the lexical value of this literal into a Timedelta. The lexical value of this literal should be in the lexical space of the duration datatype ("<http://www.w3.org/2001/XMLSchema#duration>").

**Returns**

A Timedelta value that is represented by this literal.

**is\_time()** → bool

Whether this literal is typed as time.

**parse\_time()** → datetime.time

Parses the lexical value of this literal into time. The lexical value of this literal should be in the lexical space of the time datatype ("<http://www.w3.org/2001/XMLSchema#time>").

**Returns**

A time value that is represented by this literal.

**is\_gyearmonth()** → bool

Whether this literal is typed as gYearMonth.

**parse\_gyearmonth()** → tuple

Parses the lexical value of this literal into gYearMonth.

**Returns**

A tuple value of length 2 that is represented by this literal.

**is\_gmonthday()** → bool

Whether this literal is typed as gMonthDay.

**parse\_gmonthday()** → tuple

Parses the lexical value of this literal into gMonthDay.

**Returns**

A tuple value of length 2 that is represented by this literal.

**is\_gyear()** → bool

Whether this literal is typed as gYear.

**parse\_gyear()** → tuple

Parses the lexical value of this literal into gYear.

**Returns**

A integer value that is represented by this literal.

**is\_gmonth()** → bool

Whether this literal is typed as gMonth.

**parse\_gmonth()** → tuple

Parses the lexical value of this literal into gMonth.

**Returns**

A integer value that is represented by this literal.

**is\_gday()** → bool

Whether this literal is typed as gDay.

**parse\_gday()** → tuple

Parses the lexical value of this literal into gDay.

**Returns**

A integer value that is represented by this literal.

**has\_float\_special\_value()** → bool

Whether this literal is using a float special value i.e.  $v \in [\text{"NaN"}, \text{"INF"}, \text{"-INF"}]$ , defined by and enumeration class (not pure string value).

**is\_literal()** → bool

**Returns**

true if the annotation value is a literal

**as\_literal()** → *OWLLiteral*

**Returns**

if the value is a literal, returns it. Return None otherwise

**to\_python()** → Literals

**abstract get\_datatype()** → *owlapy.owl\_datatype.OWLDatatype*

Gets the OWLDatatype which types this literal.

**Returns**

The OWLDatatype that types this literal.

## owlapy.owl\_object

OWL Base classes

### Classes

<i>OWLObject</i>	Base interface for OWL objects
<i>OWLObjectRenderer</i>	Abstract class with a render method to render an OWL Object into a string.
<i>OWLObjectParser</i>	Abstract class with a parse method to parse a string to an OWL Object.
<i>OWLNamedObject</i>	Represents a named object for example, class, property, ontology etc. - i.e. anything that has an
<i>OWLEntity</i>	Represents Entities in the OWL 2 Specification.

### Module Contents

**class** owlapy.owl\_object.OWLObject

Base interface for OWL objects

**\_\_slots\_\_** = ()

**abstract** **\_\_eq\_\_**(other)

**abstract** **\_\_hash\_\_**()

**abstract** **\_\_repr\_\_**()

**is\_anonymous**() → bool

**class** owlapy.owl\_object.OWLObjectRenderer

Abstract class with a render method to render an OWL Object into a string.

**abstract set\_short\_form\_provider** (*short\_form\_provider*) → None

Configure a short form provider that shortens the OWL objects during rendering.

**Parameters**

**short\_form\_provider** – Short form provider.

**abstract render** (*o: OWLObject*) → str

Render OWL Object to string.

**Parameters**

*o* – OWL Object.

**Returns**

String rendition of OWL object.

**class** owlapy.owl\_object.**OWLObjectParser**

Abstract class with a parse method to parse a string to an OWL Object.

**abstract parse\_expression** (*expression\_str: str*) → *OWLObject*

Parse a string to an OWL Object.

**Parameters**

**expression\_str** (*str*) – Expression string.

**Returns**

The OWL Object which is represented by the string.

**class** owlapy.owl\_object.**OWLNamedObject**

Bases: *OWLObject*, *owlapy.meta\_classes.HasIRI*

Represents a named object for example, class, property, ontology etc. - i.e. anything that has an IRI as its name.

**\_\_slots\_\_** = ()

**\_\_eq\_\_** (*other*)

**\_\_lt\_\_** (*other*)

**\_\_hash\_\_** ()

**\_\_repr\_\_** ()

**class** owlapy.owl\_object.**OWLEntity**

Bases: *OWLNamedObject*

Represents Entities in the OWL 2 Specification.

**\_\_slots\_\_** = ()

**to\_string\_id** () → str

**is\_anonymous** () → bool

## owlapy.owl\_ontology

OWL Ontology

## Attributes

<code>logger</code>
<code>OWLREADY2_FACET_KEYS</code>

## Classes

<code>OWLOntologyID</code>	An object that identifies an ontology. Since OWL 2, ontologies do not have to have an ontology IRI, or if they
<code>Ontology</code>	Represents an OWL 2 Ontology in the OWL 2 specification.
<code>SyncOntology</code>	Represents an OWL 2 Ontology in the OWL 2 specification.
<code>ToOwlready2</code>	
<code>FromOwlready2</code>	Map owlready2 classes to owlapy model classes.

## Module Contents

`owlapy.owl_ontology.logger`

```
class owlapy.owl_ontology.OWLOntologyID(ontology_iri: owlapy.iri.IRI | None = None,  
                                           version_iri: owlapy.iri.IRI | None = None)
```

An object that identifies an ontology. Since OWL 2, ontologies do not have to have an ontology IRI, or if they have an ontology IRI then they can optionally also have a version IRI. Instances of this OWLOntologyID class bundle identifying information of an ontology together. If an ontology doesn't have an ontology IRI then we say that it is "anonymous".

```
__slots__ = ('_ontology_iri', '_version_iri')
```

```
get_ontology_iri() → owlapy.iri.IRI | None
```

Gets the ontology IRI.

### Returns

Ontology IRI. If the ontology is anonymous, it will return None.

```
get_version_iri() → owlapy.iri.IRI | None
```

Gets the version IRI.

### Returns

Version IRI or None.

```
get_default_document_iri() → owlapy.iri.IRI | None
```

Gets the IRI which is used as a default for the document that contain a representation of an ontology with this ID. This will be the version IRI if there is an ontology IRI and version IRI, else it will be the ontology IRI if there is an ontology IRI but no version IRI, else it will be None if there is no ontology IRI. See Ontology Documents in the OWL 2 Structural Specification.

### Returns

the IRI that can be used as a default for an ontology document, or None.

`is_anonymous() → bool`

`__repr__()`

`__eq__(other)`

**class** `owlapy.owl_ontology.Ontology` (*manager: OM, ontology\_iri: owlapy.iri.IRI | str, load: bool*)

Bases: `owlapy.abstracts.abstract_owl_ontology.AbstractOWLontology`

Represents an OWL 2 Ontology in the OWL 2 specification.

An OWLontology consists of a possibly empty set of OWLAxioms and a possibly empty set of OWLAnnotations. An ontology can have an ontology IRI which can be used to identify the ontology. If it has an ontology IRI then it may also have an ontology version IRI. Since OWL 2, an ontology need not have an ontology IRI. (See the OWL 2 Structural Specification).

An ontology cannot be modified directly. Changes must be applied via its OWLontologyManager.

`__slots__ = ('_manager', '_iri', '_world', '_onto', 'is_modified')`

`is_modified: bool`

`__len__() → int`

`classes_in_signature() → Iterable[owlapy.class_expression.OWLClass]`

Gets the classes in the signature of this object.

**Returns**

Classes in the signature of this object.

`data_properties_in_signature() → Iterable[owlapy.owl_property.OWLDataProperty]`

Get the data properties that are in the signature of this object.

**Returns**

Data properties that are in the signature of this object.

`object_properties_in_signature() → Iterable[owlapy.owl_property.OWLObjectProperty]`

A convenience method that obtains the object properties that are in the signature of this object.

**Returns**

Object properties that are in the signature of this object.

`properties_in_signature() → Iterable[owlapy.owl_property.OWLProperty]`

`individuals_in_signature() → Iterable[owlapy.owl_individual.OWLNamedIndividual]`

A convenience method that obtains the individuals that are in the signature of this object.

**Returns**

Individuals that are in the signature of this object.

**abstract** `get_abox_axioms() → Iterable`

**abstract** `get_tbox_axioms() → Iterable`

**abstract** `get_abox_axioms_between_individuals() → Iterable`

**abstract** `get_abox_axioms_between_individuals_and_classes() → Iterable`

**equivalent\_classes\_axioms** (*c*: *owlapy.class\_expression.OWLClass*)  
→ Iterable[*owlapy.owl\_axiom.OWLEquivalentClassesAxiom*]

Gets all of the equivalent axioms in this ontology that contain the specified class as an operand.

**Parameters**

**c** – The class for which the EquivalentClasses axioms should be retrieved.

**Returns**

EquivalentClasses axioms contained in this ontology.

**general\_class\_axioms** () → Iterable[*owlapy.owl\_axiom.OWLClassAxiom*]

**Get the general class axioms of this ontology. This includes SubClass axioms with a complex class expression**

as the sub class and EquivalentClass axioms and DisjointClass axioms with only complex class expressions.

**Returns**

General class axioms contained in this ontology.

**get\_owl\_ontology\_manager** () → *\_OM*

Gets the manager that manages this ontology.

**get\_ontology\_id** () → *OWLOntologyID*

Gets the OWLOntologyID belonging to this object.

**Returns**

The OWLOntologyID.

**data\_property\_domain\_axioms** (*pe*: *owlapy.owl\_property.OWLDataProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyDomainAxiom*]

Gets the OWLDataPropertyDomainAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**data\_property\_range\_axioms** (*pe*: *owlapy.owl\_property.OWLDataProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyRangeAxiom*]

Gets the OWLDataPropertyRangeAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**object\_property\_domain\_axioms** (*pe*: *owlapy.owl\_property.OWLObjectProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLObjectPropertyDomainAxiom*]

Gets the OWLObjectPropertyDomainAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**object\_property\_range\_axioms** (*pe: owlapy.owl\_property.OWLObjectProperty*)  
→ *Iterable[owlapy.owl\_axiom.OWLObjectPropertyRangeAxiom]*

Gets the OWLObjectPropertyRangeAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**add\_axiom** (*axiom: owlapy.owl\_axiom.OWLAxiom | Iterable[owlapy.owl\_axiom.OWLAxiom]*)

Add the specified axiom/axioms to the ontology.

**Parameters**

**axiom** – Can be a single axiom or a collection of axioms.

**Returns**

Nothing.

**remove\_axiom** (*axiom: owlapy.owl\_axiom.OWLAxiom | Iterable[owlapy.owl\_axiom.OWLAxiom]*)

Removes the specified axiom/axioms to the ontology.

**Parameters**

**axiom** – Can be a single axiom or a collection of axioms.

**Returns**

Nothing.

**save** (*path: str | owlapy.iri.IRI = None, inplace: bool = False, rdf\_format='rdxml'*)

Saves this ontology, using its IRI to determine where/how the ontology should be saved.

**Parameters**

**document\_iri** – Whether you want to save in a different location.

**get\_original\_iri** ()

Get the IRI argument that was used to create this ontology.

**\_\_eq\_\_** (*other*)

**\_\_hash\_\_** ()

**\_\_repr\_\_** ()

**class** owlapy.owl\_ontology.**SyncOntology** (*manager: \_SM, path: owlapy.iri.IRI | str,*  
*new: bool = False*)

Bases: *owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLontology*

Represents an OWL 2 Ontology in the OWL 2 specification.

An OWLontology consists of a possibly empty set of OWLAxioms and a possibly empty set of OWLAnnotations. An ontology can have an ontology IRI which can be used to identify the ontology. If it has an ontology IRI then it may also have an ontology version IRI. Since OWL 2, an ontology need not have an ontology IRI. (See the OWL 2 Structural Specification).

An ontology cannot be modified directly. Changes must be applied via its OWLontologyManager.

**manager**

**path**

**new**



**mapper**

**\_\_eq\_\_**(*other*)

**\_\_hash\_\_**()

**\_\_repr\_\_**()

**\_\_len\_\_**()

**classes\_in\_signature**() → Iterable[*owlapy.class\_expression.OWLClass*]

Gets the classes in the signature of this object.

**Returns**

Classes in the signature of this object.

**data\_properties\_in\_signature**() → Iterable[*owlapy.owl\_property.OWLDataProperty*]

Get the data properties that are in the signature of this object.

**Returns**

Data properties that are in the signature of this object.

**object\_properties\_in\_signature**() → Iterable[*owlapy.owl\_property.OWLObjectProperty*]

A convenience method that obtains the object properties that are in the signature of this object.

**Returns**

Object properties that are in the signature of this object.

**individuals\_in\_signature**() → Iterable[*owlapy.owl\_individual.OWLNamedIndividual*]

A convenience method that obtains the individuals that are in the signature of this object.

**Returns**

Individuals that are in the signature of this object.

**equivalent\_classes\_axioms**(*c: owlapy.class\_expression.OWLClass*)  
→ Iterable[*owlapy.owl\_axiom.OWLEquivalentClassesAxiom*]

Gets all of the equivalent axioms in this ontology that contain the specified class as an operand.

**Parameters**

**c** – The class for which the EquivalentClasses axioms should be retrieved.

**Returns**

EquivalentClasses axioms contained in this ontology.

**general\_class\_axioms**() → Iterable[*owlapy.owl\_axiom.OWLClassAxiom*]

**Get the general class axioms of this ontology. This includes SubClass axioms with a complex class expression**

as the sub class and EquivalentClass axioms and DisjointClass axioms with only complex class expressions.

**Returns**

General class axioms contained in this ontology.

**data\_property\_domain\_axioms**(*property: owlapy.owl\_property.OWLDataProperty*)  
→ Iterable[*owlapy.owl\_axiom.OWLDataPropertyDomainAxiom*]

Gets the OWLDataPropertyDomainAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**data\_property\_range\_axioms** (*property: owlapy.owl\_property.OWLDataProperty*)  
 → Iterable[*owlapy.owl\_axiom.OWLDataPropertyRangeAxiom*]

Gets the OWLDataPropertyRangeAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**object\_property\_domain\_axioms** (*property: owlapy.owl\_property.OWLObjectProperty*)  
 → Iterable[*owlapy.owl\_axiom.OWLObjectPropertyDomainAxiom*]

Gets the OWLObjectPropertyDomainAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**object\_property\_range\_axioms** (*property: owlapy.owl\_property.OWLObjectProperty*)  
 → Iterable[*owlapy.owl\_axiom.OWLObjectPropertyRangeAxiom*]

Gets the OWLObjectPropertyRangeAxiom objects where the property is equal to the specified property.

**Parameters**

**property** – The property which is equal to the property of the retrieved axioms.

**Returns**

The axioms matching the search.

**get\_signature** (*include\_imports\_closure: bool = True*)

Gets the entities that are in the signature of this ontology.

**Parameters**

**include\_imports\_closure** – Whether to include/exclude imports from searches.

**Returns**

Entities in signature.

**get\_abox\_axioms** (*include\_imports\_closure: bool = True*) → Iterable[*owlapy.owl\_axiom.OWLAxiom*]

Get all ABox axioms.

**Parameters**

**include\_imports\_closure** – Whether to include/exclude imports from searches.

**Returns**

ABox axioms.

**get\_tbox\_axioms** (*include\_imports\_closure: bool = True*) → Iterable[*owlapy.owl\_axiom.OWLAxiom*]

Get all TBox axioms.

**Parameters**

**include\_imports\_closure** – Whether to include/exclude imports from searches.

**Returns**

TBox axioms.

```

get_owl_ontology_manager() → _M
    Gets the manager that manages this ontology.

get_owlapi_ontology()

get_ontology_id() → OWLOntologyID
    Gets the OWLOntologyID belonging to this object.

    Returns
        The OWLOntologyID.

add_axiom(axiom: owlapy.owl_axiom.OWLAXiom | Iterable[owlapy.owl_axiom.OWLAXiom])
    Add the specified axiom/axioms to the ontology.

    Parameters
        axiom – Can be a single axiom or a collection of axioms.

    Returns
        Nothing.

remove_axiom(axiom: owlapy.owl_axiom.OWLAXiom | Iterable[owlapy.owl_axiom.OWLAXiom])
    Removes the specified axiom/axioms to the ontology.

    Parameters
        axiom – Can be a single axiom or a collection of axioms.

    Returns
        Nothing.

save(path: str = None, document_iri: owlapy.iri.IRI | None = None)

https://github.com/phillord/owl-api/blob/b2a5bf9a0c6730c8ff950776af8f9bf19c78eac/contract/src/test/java/org/coode/owlapi/examples/Examples.java#L206

owlapy.owl_ontology.OWLREADY2_FACET_KEYS

class owlapy.owl_ontology.ToOwlready2(world: owlready2.World)

    __slots__ = '_world'

    abstract map_object(o: owlapy.owl_object.OWLObject)
        Map owlapy object classes.

    abstract map_concept(o: owlapy.class_expression.OWLClassExpression)
        → owlready2.ClassConstruct | owlready2.ThingClass
        Map owlapy concept classes.

    abstract map_datarange(p: owlapy.owl_data_ranges.OWLDataRange)
        → owlready2.ClassConstruct | type
        Map owlapy data range classes.

class owlapy.owl_ontology.FromOwlready2
    Map owlready2 classes to owlapy model classes.

    __slots__ = ()

    abstract map_concept(c: owlready2.ClassConstruct | owlready2.ThingClass)
        → owlapy.class_expression.OWLClassExpression
        Map concept classes.

    abstract map_datarange(p: owlready2.ClassConstruct) → owlapy.owl_data_ranges.OWLDataRange
        Map data range classes.

```

## owlapy.owl\_ontology\_manager

### Classes

<i>OWLImportsDeclaration</i>	Represents an import statement in an ontology.
<i>AddImport</i>	Represents an ontology change where an import statement is added to an ontology.
<i>OntologyManager</i>	An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing
<i>SyncOntologyManager</i>	Create OWLManager in Python

### Module Contents

**class** owlapy.owl\_ontology\_manager.**OWLImportsDeclaration** (*import\_iri: owlapy.iri.IRI*)

Bases: *owlapy.meta\_classes.HasIRI*

Represents an import statement in an ontology.

**\_\_slots\_\_** = **'\_iri'**

**property** **iri:** *owlapy.iri.IRI*

Gets the import IRI.

#### Returns

The import IRI that points to the ontology to be imported. The imported ontology might have this IRI as its ontology IRI but this is not mandated. For example, an ontology with a non-resolvable ontology IRI can be deployed at a resolvable URL.

**property** **str:** **str**

Gets the string representation of this object

#### Returns

The IRI as string

**class** owlapy.owl\_ontology\_manager.**AddImport** (  
    *ontology: owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology,*  
    *import\_declaration: OWLImportsDeclaration*)

Bases: *owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyChange*

Represents an ontology change where an import statement is added to an ontology.

**\_\_slots\_\_** = (**'\_ont'**, **'\_declaration'**)

**get\_import\_declaration**() → *OWLImportsDeclaration*

Gets the import declaration that the change pertains to.

#### Returns

The import declaration.

**class** owlapy.owl\_ontology\_manager.**OntologyManager** (*world\_store=None*)

Bases: *owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyManager*

An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing ontologies.

**\_\_slots\_\_** = **'\_world'**

**create\_ontology** (*iri: str | owlapy.iri.IRI = None*) → *owlapy.owl\_ontology.Ontology*

Creates a new (empty) ontology that has the specified ontology IRI (and no version IRI).

**Parameters**

**iri** – The IRI of the ontology to be created, can also be a string.

**Returns**

The newly created ontology.

**load\_ontology** (*path: str = None*) → *owlapy.owl\_ontology.Ontology*

Loads an ontology that is assumed to have the specified ontology IRI as its IRI or version IRI. The ontology IRI will be mapped to an ontology document IRI.

**Parameters**

**iri** –

**The IRI that identifies the ontology, can also be a string.**

It is expected that the ontology will also have this IRI

(although the OWL API should tolerate situations where this is not the case).

**Returns**

The OWLOntology representation of the ontology that was loaded.

**apply\_change** (*change: owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLontologyChange*)

A convenience method that applies just one change to an ontology. When this method is used through an OWLOntologyManager implementation, the instance used should be the one that the ontology returns through the get\_owl\_ontology\_manager() call.

**Parameters**

**change** – The change to be applied.

**Raises**

**ChangeApplied.UNSUCCESSFULLY** – if the change was not applied successfully.

**save\_world()**

Saves the actual state of the quadstore in the SQLite3 file.

**class** owlapy.owl\_ontology\_manager.**SyncOntologyManager**

Bases: *owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLontologyManager*

Create OWLManager in Python [https://owllcs.github.io/owlapi/apidocs\\_5/org/semanticweb/owlapi/apibinding/OWLManager.html](https://owllcs.github.io/owlapi/apidocs_5/org/semanticweb/owlapi/apibinding/OWLManager.html)

**owlapi\_manager**

**create\_ontology** (*iri: owlapy.iri.IRI | str*) → *owlapy.owl\_ontology.SyncOntology*

Creates a new (empty) ontology that has the specified ontology IRI (and no version IRI).

**Parameters**

**iri** – The IRI of the ontology to be created, can also be a string.

**Returns**

The newly created ontology.

**load\_ontology** (*path: str*) → *owlapy.owl\_ontology.SyncOntology*

Loads an ontology that is assumed to have the specified ontology IRI as its IRI or version IRI. The ontology IRI will be mapped to an ontology document IRI.

**Parameters**

**iri** –

**The IRI that identifies the ontology, can also be a string.**

It is expected that the ontology will also have this IRI

(although the OWL API should tolerate situations where this is not the case).

#### Returns

The OWLOntology representation of the ontology that was loaded.

`get_owlapi_manager()`

`abstract apply_change (`

*change: owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyChange)*

A convenience method that applies just one change to an ontology. When this method is used through an OWLOntologyManager implementation, the instance used should be the one that the ontology returns through the `get_owl_ontology_manager()` call.

#### Parameters

**change** – The change to be applied.

#### Raises

**ChangeApplied.UNSUCCESSFULLY** – if the change was not applied successfully.

`getOntologyFormat (*args)`

`saveOntology (*args) → None`

## owlapy.owl\_property

OWL Properties

### Classes

<i>OWLPropertyExpression</i>	Represents a property or possibly the inverse of a property.
<i>OWLObjectPropertyExpression</i>	A high level interface to describe different types of object properties.
<i>OWLDataPropertyExpression</i>	A high level interface to describe different types of data properties.
<i>OWLProperty</i>	A base class for properties that aren't expression i.e. named properties. By definition, properties
<i>OWLObjectProperty</i>	Represents an Object Property in the OWL 2 Specification. Object properties connect pairs of individuals.
<i>OWLObjectInverseOf</i>	Represents the inverse of a property expression (Object-InverseOf). An inverse object property expression
<i>OWLDataProperty</i>	Represents a Data Property in the OWL 2 Specification. Data properties connect individuals with literals.

### Module Contents

**class** owlapy.owl\_property.OWLPropertyExpression

Bases: *owlapy.owl\_object.OWLObject*

Represents a property or possibly the inverse of a property.

**\_\_slots\_\_** = ()

**is\_data\_property\_expression()** → bool

**Returns**

True if this is a data property.

**is\_object\_property\_expression()** → bool

**Returns**

True if this is an object property.

**is\_owl\_top\_object\_property()** → bool

Determines if this is the owl:topObjectProperty.

**Returns**

topObjectProperty.

**Return type**

True if this property is the owl

**is\_owl\_top\_data\_property()** → bool

Determines if this is the owl:topDataProperty.

**Returns**

topDataProperty.

**Return type**

True if this property is the owl

**class** owlapy.owl\_property.OWLObjectPropertyExpression

Bases: *OWLPropertyExpression*

A high level interface to describe different types of object properties.

**\_\_slots\_\_** = ()

**abstract get\_inverse\_property()** → *OWLObjectPropertyExpression*

Obtains the property that corresponds to the inverse of this property.

**Returns**

The inverse of this property. Note that this property will not necessarily be in the simplest form.

**abstract get\_named\_property()** → *OWLObjectProperty*

Get the named object property used in this property expression.

**Returns**

P if this expression is either inv(P) or P.

**is\_object\_property\_expression()** → bool

**Returns**

True if this is an object property.

**class** owlapy.owl\_property.OWLDataPropertyExpression

Bases: *OWLPropertyExpression*

A high level interface to describe different types of data properties.

**\_\_slots\_\_** = ()

`is_data_property_expression()`

**Returns**

True if this is a data property.

`class owlapy.owl_property.OWLProperty (iri: owlapy.iri.IRI | str)`

Bases: *OWLPropertyExpression*, *owlapy.owl\_object.OWLEntity*

A base class for properties that aren't expression i.e. named properties. By definition, properties are either data properties or object properties.

`__slots__ = '_iri'`

`property str: str`

Gets the string representation of this object

**Returns**

The IRI as string

`property iri: owlapy.iri.IRI`

Gets the IRI of this object.

**Returns**

The IRI of this object.

`class owlapy.owl_property.OWLObjectProperty (iri: owlapy.iri.IRI | str)`

Bases: *OWLObjectPropertyExpression*, *OWLProperty*

Represents an Object Property in the OWL 2 Specification. Object properties connect pairs of individuals.

([https://www.w3.org/TR/owl2-syntax/#Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Object_Properties))

`__slots__ = '_iri'`

`type_index: Final = 1002`

`property reminder`

`get_named_property() → OWLObjectProperty`

Get the named object property used in this property expression.

**Returns**

P if this expression is either inv(P) or P.

`get_inverse_property() → OWLObjectInverseOf`

Obtains the property that corresponds to the inverse of this property.

**Returns**

The inverse of this property. Note that this property will not necessarily be in the simplest form.

`is_owl_top_object_property() → bool`

Determines if this is the owl:topObjectProperty.

**Returns**

topObjectProperty.

**Return type**

True if this property is the owl



```
class owlapy.owl_property.OWLObjectInverseOf (property: OWLObjectProperty)
```

Bases: *OWLObjectPropertyExpression*

Represents the inverse of a property expression (ObjectInverseOf). An inverse object property expression ObjectInverseOf( P ) connects an individual I1 with I2 if and only if the object property P connects I2 with I1. This can be used to refer to the inverse of a property, without actually naming the property. For example, consider the property hasPart, the inverse property of hasPart (isPartOf) can be referred to using this interface inverseOf(hasPart), which can be used in restrictions e.g. inverseOf(hasPart) some Car refers to the set of things that are part of at least one car.

([https://www.w3.org/TR/owl2-syntax/#Inverse\\_Object\\_Properties](https://www.w3.org/TR/owl2-syntax/#Inverse_Object_Properties))

```
__slots__ = '_inverse_property'
```

```
type_index: Final = 1003
```

```
get_inverse() → OWLObjectProperty
```

Gets the property expression that this is the inverse of.

#### Returns

The object property expression such that this object property expression is an inverse of it.

```
get_inverse_property() → OWLObjectProperty
```

Obtains the property that corresponds to the inverse of this property.

#### Returns

The inverse of this property. Note that this property will not necessarily be in the simplest form.

```
get_named_property() → OWLObjectProperty
```

Get the named object property used in this property expression.

#### Returns

P if this expression is either inv(P) or P.

```
__repr__()
```

```
__eq__(other)
```

```
__hash__()
```

```
class owlapy.owl_property.OWLDataProperty (iri: owlapy.iri.IRI | str)
```

Bases: *OWLDataPropertyExpression*, *OWLProperty*

Represents a Data Property in the OWL 2 Specification. Data properties connect individuals with literals. In some knowledge representation systems, functional data properties are called attributes.

([https://www.w3.org/TR/owl2-syntax/#Data\\_Properties](https://www.w3.org/TR/owl2-syntax/#Data_Properties))

```
__slots__ = '_iri'
```

```
type_index: Final = 1004
```

```
is_owl_top_data_property() → bool
```

Determines if this is the owl:topDataProperty.

#### Returns

topDataProperty.

#### Return type

True if this property is the owl

## owlapy.owl\_reasoner

OWL Reasoner

### Attributes

---

*logger*

---

### Classes

---

<i>StructuralReasoner</i>	Tries to check instances fast (but maybe incomplete).
<i>SyncReasoner</i>	An OWLReasoner reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of

---

### Functions

---

*initialize\_reasoner*(reasoner, owlapi\_ontology)

*import\_and\_include\_axioms\_generators*()

---

### Module Contents

owlapy.owl\_reasoner.logger

**class** owlapy.owl\_reasoner.StructuralReasoner (  
    *ontology: owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLontology, \*,*  
    *class\_cache: bool = True, property\_cache: bool = True, negation\_default: bool = True,*  
    *sub\_properties: bool = False)*

Bases: *owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner*

Tries to check instances fast (but maybe incomplete).

**class\_cache:** bool

**reset** ()

The reset method shall reset any cached state.

**data\_property\_domains** (*pe: owlapy.owl\_property.OWLDataProperty, direct: bool = False*)  
    → *Iterable[owlapy.class\_expression.OWLClassExpression]*

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **pe** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

### Returns

Let  $N = \text{equivalent\_classes}(\text{DataSomeValuesFrom}(\text{pe } \text{rdfs:Literal}))$ . If `direct` is `True`: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of `super_classes(DataSomeValuesFrom(pe rdfs:Literal), true)`. If `direct` is `False`: then the result of `super_classes(DataSomeValuesFrom(pe rdfs:Literal), false)` together with  $N$  if  $N$  is non-empty. (Note, `rdfs:Literal` is the top datatype).

**object\_property\_domains** (*pe*: *owlapy.owl\_property.OWLObjectProperty*, *direct*: *bool = False*)  
→ *Iterable[owlapy.class\_expression.OWLClassExpression]*

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

### Parameters

- **pe** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (`True`), or if all domains should be retrieved (`False`).

### Returns

Let  $N = \text{equivalent\_classes}(\text{ObjectSomeValuesFrom}(\text{pe owl:Thing}))$ . If `direct` is `True`: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of `super_classes(ObjectSomeValuesFrom(pe owl:Thing), true)`. If `direct` is `False`: then the result of `super_classes(ObjectSomeValuesFrom(pe owl:Thing), false)` together with  $N$  if  $N$  is non-empty.

**object\_property\_ranges** (*pe*: *owlapy.owl\_property.OWLObjectProperty*, *direct*: *bool = False*)  
→ *Iterable[owlapy.class\_expression.OWLClassExpression]*

**Gets the class expressions that are the direct or indirect ranges of this property with respect to the imports closure of the root ontology.**

### Parameters

- **pe** – The property expression whose ranges are to be retrieved.
- **direct** – Specifies if the direct ranges should be retrieved (`True`), or if all ranges should be retrieved (`False`).

### Returns

Let  $N = \text{equivalent\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe owl:Thing})))$ . If `direct` is `True`: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of `super_classes(ObjectSomeValuesFrom(ObjectInverseOf(pe owl:Thing), true)`. If `direct` is `False`: then the result of `super_classes(ObjectSomeValuesFrom(ObjectInverseOf(pe owl:Thing), false)` together with  $N$  if  $N$  is non-empty.

**equivalent\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *only\_named*: *bool = True*)  
→ *Iterable[owlapy.class\_expression.OWLClassExpression]*

**Gets the class expressions that are equivalent to the specified class expression with respect to the set of reasoner axioms.**

### Parameters

**ce** – The class expression whose equivalent classes are to be retrieved.

### Returns

All class expressions  $C$  where the root ontology imports closure entails `EquivalentClasses(ce C)`. If `ce` is not a class name (i.e. it is an anonymous class expression) and there are no such

classes C then there will be no result. If ce is unsatisfiable with respect to the set of reasoner axioms then owl:Nothing, i.e. the bottom node, will be returned.

**disjoint\_classes** (*ce: owlapy.class\_expression.OWLClassExpression, only\_named: bool = True*)  
→ Iterable[*owlapy.class\_expression.OWLClassExpression*]

Gets the class expressions that are disjoint with specified class expression with respect to the set of reasoner axioms.

**Parameters**

**ce** – The class expression whose disjoint classes are to be retrieved.

**Returns**

All class expressions D where the set of reasoner axioms entails EquivalentClasses(D Object-ComplementOf(ce)) or StrictSubClassOf(D ObjectComplementOf(ce)).

**different\_individuals** (*ind: owlapy.owl\_individual.OWLNamedIndividual*)  
→ Iterable[*owlapy.owl\_individual.OWLNamedIndividual*]

Gets the individuals that are different from the specified individual with respect to the set of reasoner axioms.

**Parameters**

**ind** – The individual whose different individuals are to be retrieved.

**Returns**

All individuals x where the set of reasoner axioms entails DifferentIndividuals(ind x).

**same\_individuals** (*ind: owlapy.owl\_individual.OWLNamedIndividual*)  
→ Iterable[*owlapy.owl\_individual.OWLNamedIndividual*]

Gets the individuals that are the same as the specified individual with respect to the set of reasoner axioms.

**Parameters**

**ind** – The individual whose same individuals are to be retrieved.

**Returns**

All individuals x where the root ontology imports closure entails SameIndividual(ind x).

**data\_property\_values** (*e: owlapy.owl\_object.OWLEntity, pe: owlapy.owl\_property.OWLDataProperty, direct: bool = True*) → Iterable[*owlapy.owl\_literal.OWLLiteral*]

Gets the data property values for the specified entity and data property expression.

**Parameters**

- **e** – The owl entity (usually an individual) that is the subject of the data property values.
- **pe** – The data property expression whose values are to be retrieved for the specified entity.

Note: Can be used to get values, for example, of ‘label’ property of owl entities such as classes and properties too (not only individuals).

**Returns**

A set of OWLLiterals containing literals such that for each literal l in the set, the set of reasoner axioms entails DataPropertyAssertion(pe ind l).

**all\_data\_property\_values** (*pe: owlapy.owl\_property.OWLDataProperty, direct: bool = True*)  
→ Iterable[*owlapy.owl\_literal.OWLLiteral*]

Gets all values for the given data property expression that appear in the knowledge base.

**Parameters**

- **pe** – The data property expression whose values are to be retrieved
- **direct** – Specifies if only the direct values of the data property pe should be retrieved (True), or if the values of sub properties of pe should be taken into account (False).

### Returns

A set of OWLLiterals containing literals such that for each literal *l* in the set, the set of reasoner axioms entails `DataPropertyAssertion(pe ind l)` for any *ind*.

**object\_property\_values** (*ind*: *owlapy.owl\_individual.OWLNamedIndividual*,  
*pe*: *owlapy.owl\_property.OWLObjectPropertyExpression*, *direct*: *bool = False*)  
→ *Iterable[owlapy.owl\_individual.OWLNamedIndividual]*

Gets the object property values for the specified individual and object property expression.

### Parameters

- **ind** – The individual that is the subject of the object property values.
- **pe** – The object property expression whose values are to be retrieved for the specified individual.

### Returns

The named individuals such that for each individual *j*, the set of reasoner axioms entails `ObjectPropertyAssertion(pe ind j)`.

**instances** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *direct*: *bool = False*, *timeout*: *int = 1000*)

Gets the individuals which are instances of the specified class expression.

### Parameters

- **ce** – The class expression whose instances are to be retrieved.
- **direct** – Specifies if the direct instances should be retrieved (True), or if all instances should be retrieved (False).
- **timeout** – Time limit in seconds until results must be returned, else empty set is returned.

### Returns

If *direct* is True, each named individual *j* where the set of reasoner axioms entails `DirectClassAssertion(ce, j)`. If *direct* is False, each named individual *j* where the set of reasoner axioms entails `ClassAssertion(ce, j)`. If *ce* is unsatisfiable with respect to the set of reasoner axioms then nothing returned.

**sub\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *direct*: *bool = False*,  
*only\_named*: *bool = True*) → *Iterable[owlapy.class\_expression.OWLClassExpression]*

Gets the set of named classes that are the strict (potentially direct) subclasses of the specified class expression with respect to the reasoner axioms.

### Parameters

- **ce** – The class expression whose strict (direct) subclasses are to be retrieved.
- **direct** – Specifies if the direct subclasses should be retrieved (True) or if the all subclasses (descendant) classes should be retrieved (False).

### Returns

If *direct* is True, each class *C* where reasoner axioms entails `DirectSubClassOf(C, ce)`. If *direct* is False, each class *C* where reasoner axioms entails `StrictSubClassOf(C, ce)`. If *ce* is equivalent to `owl:Nothing` then nothing will be returned.

**super\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *direct*: *bool = False*,  
*only\_named*: *bool = True*) → *Iterable[owlapy.class\_expression.OWLClassExpression]*

Gets the stream of named classes that are the strict (potentially direct) super classes of the specified class expression with respect to the imports closure of the root ontology.

### Parameters

- **ce** – The class expression whose strict (direct) super classes are to be retrieved.

- **direct** – Specifies if the direct super classes should be retrieved (True) or if the all super classes (ancestors) classes should be retrieved (False).

#### Returns

If direct is True, each class C where the set of reasoner axioms entails `DirectSubClassOf(ce, C)`. If direct is False, each class C where set of reasoner axioms entails `StrictSubClassOf(ce, C)`. If ce is equivalent to `owl:Thing` then nothing will be returned.

**equivalent\_object\_properties** (*op: owlapy.owl\_property.OWLObjectPropertyExpression*)  
→ `Iterable[owlapy.owl_property.OWLObjectPropertyExpression]`

Gets the simplified object properties that are equivalent to the specified object property with respect to the set of reasoner axioms.

#### Parameters

**op** – The object property whose equivalent object properties are to be retrieved.

#### Returns

All simplified object properties e where the root ontology imports closure entails `EquivalentObjectProperties(op e)`. If op is unsatisfiable with respect to the set of reasoner axioms then `owl:bottomDataProperty` will be returned.

**equivalent\_data\_properties** (*dp: owlapy.owl\_property.OWLDataProperty*)  
→ `Iterable[owlapy.owl_property.OWLDataProperty]`

Gets the data properties that are equivalent to the specified data property with respect to the set of reasoner axioms.

#### Parameters

**dp** – The data property whose equivalent data properties are to be retrieved.

#### Returns

All data properties e where the root ontology imports closure entails `EquivalentDataProperties(dp e)`. If dp is unsatisfiable with respect to the set of reasoner axioms then `owl:bottomDataProperty` will be returned.

**disjoint\_object\_properties** (*op: owlapy.owl\_property.OWLObjectPropertyExpression*)  
→ `Iterable[owlapy.owl_property.OWLObjectPropertyExpression]`

Gets the simplified object properties that are disjoint with the specified object property with respect to the set of reasoner axioms.

#### Parameters

**op** – The object property whose disjoint object properties are to be retrieved.

#### Returns

All simplified object properties e where the root ontology imports closure entails `EquivalentObjectProperties(e ObjectPropertyComplementOf(op))` or `StrictSubObjectPropertyOf(e ObjectPropertyComplementOf(op))`.

**disjoint\_data\_properties** (*dp: owlapy.owl\_property.OWLDataProperty*)  
→ `Iterable[owlapy.owl_property.OWLDataProperty]`

Gets the data properties that are disjoint with the specified data property with respect to the set of reasoner axioms.

#### Parameters

**dp** – The data property whose disjoint data properties are to be retrieved.

#### Returns

All data properties e where the root ontology imports closure entails `EquivalentDataProperties(e DataPropertyComplementOf(dp))` or `StrictSubDataPropertyOf(e DataPropertyComplementOf(dp))`.

**super\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*, *direct*: *bool = False*)  
→ *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the stream of data properties that are the strict (potentially direct) super properties of the specified data property with respect to the imports closure of the root ontology.

#### Parameters

- **dp** (*OWLDataProperty*) – The data property whose super properties are to be retrieved.
- **direct** (*bool*) – Specifies if the direct super properties should be retrieved (True) or if the all super properties (ancestors) should be retrieved (False).

#### Returns

Iterable of super properties.

**sub\_data\_properties** (*dp*: *owlapy.owl\_property.OWLDataProperty*, *direct*: *bool = False*)  
→ *Iterable[owlapy.owl\_property.OWLDataProperty]*

Gets the set of named data properties that are the strict (potentially direct) subproperties of the specified data property expression with respect to the imports closure of the root ontology.

#### Parameters

- **dp** – The data property whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

#### Returns

If *direct* is True, each property *P* where the set of reasoner axioms entails *DirectSubDataPropertyOf(P, pe)*. If *direct* is False, each property *P* where the set of reasoner axioms entails *StrictSubDataPropertyOf(P, pe)*. If *pe* is equivalent to *owl:bottomDataProperty* then nothing will be returned.

**super\_object\_properties** (*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*,  
*direct*: *bool = False*) → *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the stream of object properties that are the strict (potentially direct) super properties of the specified object property with respect to the imports closure of the root ontology.

#### Parameters

- **op** (*OWLObjectPropertyExpression*) – The object property expression whose super properties are to be retrieved.
- **direct** (*bool*) – Specifies if the direct super properties should be retrieved (True) or if the all super properties (ancestors) should be retrieved (False).

#### Returns

Iterable of super properties.

**sub\_object\_properties** (*op*: *owlapy.owl\_property.OWLObjectPropertyExpression*, *direct*: *bool = False*)  
→ *Iterable[owlapy.owl\_property.OWLObjectPropertyExpression]*

Gets the stream of simplified object property expressions that are the strict (potentially direct) subproperties of the specified object property expression with respect to the imports closure of the root ontology.

#### Parameters

- **op** – The object property expression whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

### Returns

If `direct` is `True`, simplified object property expressions, such that for each simplified object property expression, `P`, the set of reasoner axioms entails `DirectSubObjectPropertyOf(P, pe)`. If `direct` is `False`, simplified object property expressions, such that for each simplified object property expression, `P`, the set of reasoner axioms entails `StrictSubObjectPropertyOf(P, pe)`. If `pe` is equivalent to `owl:bottomObjectProperty` then nothing will be returned.

**types** (*ind*: `owlapy.owl_individual.OWLNamedIndividual`, *direct*: `bool = False`)  
→ `Iterable[owlapy.class_expression.OWLClass]`

Gets the named classes which are (potentially direct) types of the specified named individual.

### Parameters

- **ind** – The individual whose types are to be retrieved.
- **direct** – Specifies if the direct types should be retrieved (`True`), or if all types should be retrieved (`False`).

### Returns

If `direct` is `True`, each named class `C` where the set of reasoner axioms entails `DirectClassAssertion(C, ind)`. If `direct` is `False`, each named class `C` where the set of reasoner axioms entails `ClassAssertion(C, ind)`.

**get\_root\_ontology** () → `owlapy.abstracts.abstract_owl_ontology.AbstractOWL ontology`

Gets the “root” ontology that is loaded into this reasoner. The reasoner takes into account the axioms in this ontology and its import’s closure.

**get\_instances\_from\_owl\_class** (*c*: `owlapy.class_expression.OWLClass`)

**reset\_and\_disable\_cache** ()

**class** `owlapy.owl_reasoner.SyncReasoner` (*ontology*: `owlapy.owl_ontology.SyncOntology` | *str*,  
*reasoner*=`‘HermiT’`)

Bases: `owlapy.abstracts.abstract_owl_reasoner.AbstractOWLReasoner`

An `OWLReasoner` reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of a particular ontology - the “root” ontology.

**mapper**

**inference\_types\_mapping**

**instances** (*ce*: `owlapy.class_expression.OWLClassExpression`, *direct*: `bool = False`, *timeout*: `int = 1000`)

Gets the individuals which are instances of the specified class expression.

### Parameters

- **ce** – The class expression whose instances are to be retrieved.
- **direct** – Specifies if the direct instances should be retrieved (`True`), or if all instances should be retrieved (`False`).
- **timeout** – Time limit in seconds until results must be returned, else empty set is returned.

### Returns

If `direct` is `True`, each named individual `j` where the set of reasoner axioms entails `DirectClassAssertion(ce, j)`. If `direct` is `False`, each named individual `j` where the set of reasoner axioms entails `ClassAssertion(ce, j)`. If `ce` is unsatisfiable with respect to the set of reasoner axioms then nothing returned.



**equivalent\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*)  
→ List[*owlapy.class\_expression.OWLClassExpression*]

Gets the set of named classes that are equivalent to the specified class expression with respect to the set of reasoner axioms.

**Parameters**

**ce** (*OWLClassExpression*) – The class expression whose equivalent classes are to be retrieved.

**Returns**

Equivalent classes of the given class expression.

**disjoint\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*)  
→ List[*owlapy.class\_expression.OWLClassExpression*]

Gets the classes that are disjoint with the specified class expression.

**Parameters**

**ce** (*OWLClassExpression*) – The class expression whose disjoint classes are to be retrieved.

**Returns**

Disjoint classes of the given class expression.

**sub\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *direct=False*)  
→ List[*owlapy.class\_expression.OWLClassExpression*]

Gets the set of named classes that are the strict (potentially direct) subclasses of the specified class expression with respect to the reasoner axioms.

**Args:**

**ce** (*OWLClassExpression*): The class expression whose strict (direct) subclasses are to be retrieved. **direct** (*bool*, *optional*): Specifies if the direct subclasses should be retrieved (*True*) or if

all subclasses (descendant) classes should be retrieved (*False*). Defaults to *False*.

**Returns**

The subclasses of the given class expression depending on *direct* field.

**super\_classes** (*ce*: *owlapy.class\_expression.OWLClassExpression*, *direct=False*)  
→ List[*owlapy.class\_expression.OWLClassExpression*]

Gets the stream of named classes that are the strict (potentially direct) super classes of the specified class expression with respect to the imports closure of the root ontology.

**Parameters**

- **ce** (*OWLClassExpression*) – The class expression whose strict (direct) subclasses are to be retrieved.
- **direct** (*bool*, *optional*) – Specifies if the direct superclasses should be retrieved (*True*) or if all superclasses (descendant) classes should be retrieved (*False*). Defaults to *False*.

**Returns**

The subclasses of the given class expression depending on *direct* field.

**data\_property\_domains** (*p*: *owlapy.owl\_property.OWLDataProperty*, *direct: bool = False*)

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

**Parameters**

- **p** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

#### Returns

Let  $N = \text{equivalent\_classes}(\text{DataSomeValuesFrom}(\text{pe } \text{rdfs:Literal}))$ . If **direct** is True: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of  $\text{super\_classes}(\text{DataSomeValuesFrom}(\text{pe } \text{rdfs:Literal}), \text{true})$ . If **direct** is False: then the result of  $\text{super\_classes}(\text{DataSomeValuesFrom}(\text{pe } \text{rdfs:Literal}), \text{false})$  together with  $N$  if  $N$  is non-empty. (Note,  $\text{rdfs:Literal}$  is the top datatype).

**object\_property\_domains** (*p: owlapy.owl\_property.OWLObjectProperty, direct: bool = False*)

**Gets the class expressions that are the direct or indirect domains of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **p** – The property expression whose domains are to be retrieved.
- **direct** – Specifies if the direct domains should be retrieved (True), or if all domains should be retrieved (False).

#### Returns

Let  $N = \text{equivalent\_classes}(\text{ObjectSomeValuesFrom}(\text{pe } \text{owl:Thing}))$ . If **direct** is True: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{pe } \text{owl:Thing}), \text{true})$ . If **direct** is False: then the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{pe } \text{owl:Thing}), \text{false})$  together with  $N$  if  $N$  is non-empty.

**object\_property\_ranges** (*p: owlapy.owl\_property.OWLObjectProperty, direct: bool = False*)

**Gets the class expressions that are the direct or indirect ranges of this property with respect to the imports closure of the root ontology.**

#### Parameters

- **p** – The property expression whose ranges are to be retrieved.
- **direct** – Specifies if the direct ranges should be retrieved (True), or if all ranges should be retrieved (False).

#### Returns

Let  $N = \text{equivalent\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe } \text{owl:Thing})))$ . If **direct** is True: then if  $N$  is not empty then the return value is  $N$ , else the return value is the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe } \text{owl:Thing})), \text{true})$ . If **direct** is False: then the result of  $\text{super\_classes}(\text{ObjectSomeValuesFrom}(\text{ObjectInverseOf}(\text{pe } \text{owl:Thing})), \text{false})$  together with  $N$  if  $N$  is non-empty.

**sub\_object\_properties** (*p: owlapy.owl\_property.OWLObjectProperty, direct: bool = False*)

**Gets the stream of simplified object property expressions that are the strict (potentially direct) subproperties of the specified object property expression with respect to the imports closure of the root ontology.**

#### Parameters

- **p** – The object property expression whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (True) or if the all subproperties (descendants) should be retrieved (False).

### Returns

If `direct` is `True`, simplified object property expressions, such that for each simplified object property expression, `P`, the set of reasoner axioms entails `DirectSubObjectPropertyOf(P, pe)`. If `direct` is `False`, simplified object property expressions, such that for each simplified object property expression, `P`, the set of reasoner axioms entails `StrictSubObjectPropertyOf(P, pe)`. If `pe` is equivalent to `owl:bottomObjectProperty` then nothing will be returned.

**super\_object\_properties** (*p*: *owlapy.owl\_property.OWLObjectProperty*, *direct*: *bool = False*)

Gets the stream of object properties that are the strict (potentially direct) super properties of the specified object property with respect to the imports closure of the root ontology.

### Parameters

- **p** (*OWLObjectPropertyExpression*) – The object property expression whose super properties are to be retrieved.
- **direct** (*bool*) – Specifies if the direct super properties should be retrieved (`True`) or if the all super properties (ancestors) should be retrieved (`False`).

### Returns

Iterable of super properties.

**sub\_data\_properties** (*p*: *owlapy.owl\_property.OWLDataProperty*, *direct*: *bool = False*)

Gets the set of named data properties that are the strict (potentially direct) subproperties of the specified data property expression with respect to the imports closure of the root ontology.

### Parameters

- **p** – The data property whose strict (direct) subproperties are to be retrieved.
- **direct** – Specifies if the direct subproperties should be retrieved (`True`) or if the all subproperties (descendants) should be retrieved (`False`).

### Returns

If `direct` is `True`, each property `P` where the set of reasoner axioms entails `DirectSubDataPropertyOf(P, pe)`. If `direct` is `False`, each property `P` where the set of reasoner axioms entails `StrictSubDataPropertyOf(P, pe)`. If `pe` is equivalent to `owl:bottomDataProperty` then nothing will be returned.

**super\_data\_properties** (*p*: *owlapy.owl\_property.OWLDataProperty*, *direct*: *bool = False*)

Gets the stream of data properties that are the strict (potentially direct) super properties of the specified data property with respect to the imports closure of the root ontology.

### Parameters

- **p** (*OWLDataProperty*) – The data property whose super properties are to be retrieved.
- **direct** (*bool*) – Specifies if the direct super properties should be retrieved (`True`) or if the all super properties (ancestors) should be retrieved (`False`).

### Returns

Iterable of super properties.

**different\_individuals** (*i*: *owlapy.owl\_individual.OWLNamedIndividual*)

Gets the individuals that are different from the specified individual with respect to the set of reasoner axioms.

### Parameters

- **i** – The individual whose different individuals are to be retrieved.

### Returns

All individuals `x` where the set of reasoner axioms entails `DifferentIndividuals(ind x)`.

**same\_individuals** (*i*: *owlapy.owl\_individual.OWLNamedIndividual*)

Gets the individuals that are the same as the specified individual with respect to the set of reasoner axioms.

**Parameters**

**i** – The individual whose same individuals are to be retrieved.

**Returns**

All individuals *x* where the root ontology imports closure entails SameIndividual(ind *x*).

**equivalent\_object\_properties** (*p*: *owlapy.owl\_property.OWLObjectProperty*)

Gets the simplified object properties that are equivalent to the specified object property with respect to the set of reasoner axioms.

**Parameters**

**p** – The object property whose equivalent object properties are to be retrieved.

**Returns**

All simplified object properties *e* where the root ontology imports closure entails EquivalentObjectProperties(op *e*). If *op* is unsatisfiable with respect to the set of reasoner axioms then owl:bottomDataProperty will be returned.

**equivalent\_data\_properties** (*p*: *owlapy.owl\_property.OWLDataProperty*)

Gets the data properties that are equivalent to the specified data property with respect to the set of reasoner axioms.

**Parameters**

**p** – The data property whose equivalent data properties are to be retrieved.

**Returns**

All data properties *e* where the root ontology imports closure entails EquivalentDataProperties(dp *e*). If *dp* is unsatisfiable with respect to the set of reasoner axioms then owl:bottomDataProperty will be returned.

**object\_property\_values** (*i*: *owlapy.owl\_individual.OWLNamedIndividual*,  
*p*: *owlapy.owl\_property.OWLObjectProperty*)

Gets the object property values for the specified individual and object property expression.

**Parameters**

- **i** – The individual that is the subject of the object property values.
- **p** – The object property expression whose values are to be retrieved for the specified individual.

**Returns**

The named individuals such that for each individual *j*, the set of reasoner axioms entails ObjectPropertyAssertion(pe ind *j*).

**data\_property\_values** (*e*: *owlapy.owl\_object.OWLEntity*, *p*: *owlapy.owl\_property.OWLDataProperty*)

Gets the data property values for the specified entity and data property expression.

**Parameters**

- **e** – The entity (usually an individual) that is the subject of the data property values.
- **p** – The data property expression whose values are to be retrieved for the specified individual.

**Returns**

A set of OWLLiterals containing literals such that for each literal *l* in the set, the set of reasoner axioms entails DataPropertyAssertion(pe ind *l*).

**disjoint\_object\_properties** (*p*: *owlapy.owl\_property.OWLObjectProperty*)

Gets the simplified object properties that are disjoint with the specified object property with respect to the set of reasoner axioms.

**Parameters**

**p** – The object property whose disjoint object properties are to be retrieved.

**Returns**

All simplified object properties *e* where the root ontology imports closure entails `EquivalentObjectProperties(e ObjectPropertyComplementOf(op))` or `StrictSubObjectPropertyOf(e ObjectPropertyComplementOf(op))`.

**disjoint\_data\_properties** (*p*: *owlapy.owl\_property.OWLDataProperty*)

Gets the data properties that are disjoint with the specified data property with respect to the set of reasoner axioms.

**Parameters**

**p** – The data property whose disjoint data properties are to be retrieved.

**Returns**

All data properties *e* where the root ontology imports closure entails `EquivalentDataProperties(e DataPropertyComplementOf(dp))` or `StrictSubDataPropertyOf(e DataPropertyComplementOf(dp))`.

**types** (*individual*: *owlapy.owl\_individual.OWLNamedIndividual*, *direct*: *bool = False*)

Gets the named classes which are (potentially direct) types of the specified named individual.

**Parameters**

- **individual** – The individual whose types are to be retrieved.
- **direct** – Specifies if the direct types should be retrieved (True), or if all types should be retrieved (False).

**Returns**

If *direct* is True, each named class *C* where the set of reasoner axioms entails `DirectClassAssertion(C, ind)`. If *direct* is False, each named class *C* where the set of reasoner axioms entails `ClassAssertion(C, ind)`.

**has\_consistent\_ontology** () → bool

Check if the used ontology is consistent.

**Returns**

True if the ontology used by this reasoner is consistent, False otherwise.

**Return type**

bool

**infer\_axioms** (*inference\_types*: *list[str]*) → *Iterable[owlapy.owl\_axiom.OWLAxiom]*

Infer the specified inference type of axioms for the ontology managed by this instance's reasoner and return them.

**Parameters**

**inference\_types** – Axiom inference types: Available options (can set more than 1): [`"InferredClassAssertionAxiomGenerator"`, `"InferredSubClassAxiomGenerator"`, `"InferredDisjointClassesAxiomGenerator"`, `"InferredEquivalentClassAxiomGenerator"`, `"InferredEquivalentDataPropertiesAxiomGenerator"`, `"InferredEquivalentObjectPropertyAxiomGenerator"`, `"InferredInverseObjectPropertiesAxiomGenerator"`, `"InferredSubDataPropertyAxiomGenerator"`, `"InferredSubObjectPropertyAxiomGenerator"`, `"InferredDataPropertyCharacteristicAxiomGenerator"`, `"InferredObjectPropertyCharacteristicAxiomGenerator"`]

### Returns

Iterable of inferred axioms.

**infer\_axioms\_and\_save** (*output\_path*: str = None, *output\_format*: str = None, *inference\_types*: list[str] = None)

Generates inferred axioms for the ontology managed by this instance's reasoner and saves them to a file. This function uses the OWL API to generate inferred class assertion axioms based on the ontology and reasoner associated with this instance. The inferred axioms are saved to the specified output file in the desired format.

### Parameters

- **output\_path** – The name of the file where the inferred axioms will be saved.
- **output\_format** – The format in which to save the inferred axioms. Supported formats are: - "ttl" or "turtle" for Turtle format - "rdf/xml" for RDF/XML format - "owl/xml" for OWL/XML format If not specified, the format of the original ontology is used.
- **inference\_types** – Axiom inference types: Available options (can set more than 1): ["InferredClassAssertionAxiomGenerator", "InferredSubClassAxiomGenerator", "InferredDisjointClassesAxiomGenerator", "InferredEquivalentClassAxiomGenerator", "InferredEquivalentDataPropertiesAxiomGenerator", "InferredEquivalentObjectPropertyAxiomGenerator", "InferredInverseObjectPropertiesAxiomGenerator", "InferredSubDataPropertyAxiomGenerator", "InferredSubObjectPropertyAxiomGenerator", "InferredDataPropertyCharacteristicAxiomGenerator", "InferredObjectPropertyCharacteristicAxiomGenerator"]

### Returns

None (the file is saved to the specified directory)

**generate\_and\_save\_inferred\_class\_assertion\_axioms** (*output*='temp.ttl', *output\_format*: str = None)

Generates inferred class assertion axioms for the ontology managed by this instance's reasoner and saves them to a file. This function uses the OWL API to generate inferred class assertion axioms based on the ontology and reasoner associated with this instance. The inferred axioms are saved to the specified output file in the desired format. Parameters: ——— output : str, optional

The name of the file where the inferred axioms will be saved. Default is "temp.ttl".

### output\_format

[str, optional] The format in which to save the inferred axioms. Supported formats are: - "ttl" or "turtle" for Turtle format - "rdf/xml" for RDF/XML format - "owl/xml" for OWL/XML format If not specified, the format of the original ontology is used.

### Notes:

- The function supports saving in multiple formats: Turtle, RDF/XML, and OWL/XML.
- The inferred axioms are generated using the reasoner associated with this instance and the OWL API's InferredClassAssertionAxiomGenerator.
- The inferred axioms are added to a new ontology which is then saved in the specified format.

### Example:

```
>>> instance.generate_and_save_inferred_class_assertion_axioms(output=  
↳ "inferred_axioms.ttl", format="ttl")  
This will save the inferred class assertion axioms to the file "inferred_  
↳ axioms.ttl" in Turtle format.
```

**is\_entailed** (*axiom*: *owlapy.owl\_axiom.OWLAxiom*) → bool

A convenience method that determines if the specified axiom is entailed by the set of reasoner axioms.

**Parameters**

**axiom** – The axiom to check for entailment.

**Returns**

True if the axiom is entailed by the reasoner axioms and False otherwise.

**is\_satisfiable** (*ce*: *owlapy.class\_expression.OWLClassExpression*) → bool

A convenience method that determines if the specified class expression is satisfiable with respect to the reasoner axioms.

**Parameters**

**ce** – The class expression to check for satisfiability.

**Returns**

True if the class expression is satisfiable by the reasoner axioms and False otherwise.

**unsatisfiable\_classes** ()

A convenience method that obtains the classes in the signature of the root ontology that are unsatisfiable.

**get\_root\_ontology** () → *owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLontology*

Gets the “root” ontology that is loaded into this reasoner. The reasoner takes into account the axioms in this ontology and its import’s closure.

*owlapy.owl\_reasoner.initialize\_reasoner* (*reasoner*: str, *owlapi\_ontology*)

*owlapy.owl\_reasoner.import\_and\_include\_axioms\_generators* ()

## owlapy.owlapi\_mapper

### Classes

<i>OWLAPIMapper</i>	A bridge between owlapy and owlapi owl-related classes.
---------------------	---

### Functions

<i>init</i> ( <i>the_class</i> )	Since classes names in owlapi and owlapy are pretty much similar with the small difference that in owlapi they
----------------------------------	--

### Module Contents

*owlapy.owlapi\_mapper.init* (*the\_class*)

Since classes names in owlapi and owlapy are pretty much similar with the small difference that in owlapi they usually have the ‘Impl’ part then we can create the mapping class name dynamically reducing the amount of code significantly. That’s what this method does.

**class** *owlapy.owlapi\_mapper.OWLAPIMapper*

A bridge between owlapy and owlapi owl-related classes.

**map\_** (*e*)

(owlapy ↔ owlapi) entity mapping.

### Parameters

**e** – OWL entity/expression.

**static to\_list** (*stream\_obj*)

Converts Java Stream object to Python list

## owlapy.parser

String to OWL parsers.

### Attributes

*MANCHESTER\_GRAMMAR*

*DL\_GRAMMAR*

*DLparser*

*ManchesterParser*

### Classes

*ManchesterOWLSyntaxParser*

Manchester Syntax parser to parse strings to OWLClass-Expressions.

*DLSyntaxParser*

Description Logic Syntax parser to parse strings to OWL-ClassExpressions.

### Functions

*dl\_to\_owl\_expression*(dl\_expression, namespace)

*manchester\_to\_owl\_expression*(manchester\_expres  
...)

### Module Contents

owlapy.parser.**MANCHESTER\_GRAMMAR**

```
class owlapy.parser.ManchesterOWLSyntaxParser(  
    namespace: str | owlapy.namespaces.Namespaces | None = None, grammar=None)
```

Bases: parsimonious.nodes.NodeVisitor, owlapy.owl\_object.OWLObjectParser

Manchester Syntax parser to parse strings to OWLClassExpressions. Following: <https://www.w3.org/TR/owl2-manchester-syntax>.

**slots** = ('ns', 'grammar')

**ns**: str | owlapy.namespaces.Namespaces | None



**grammar**

**parse\_expression** (*expression\_str*: *str*) → *owlapy.class\_expression.OWLClassExpression*

Parse a string to an OWL Object.

**Parameters**

**expression\_str** (*str*) – Expression string.

**Returns**

The OWL Object which is represented by the string.

**visit\_union** (*node*, *children*) → *owlapy.class\_expression.OWLClassExpression*

**visit\_intersection** (*node*, *children*) → *owlapy.class\_expression.OWLClassExpression*

**visit\_primary** (*node*, *children*) → *owlapy.class\_expression.OWLClassExpression*

**visit\_some\_only\_res** (*node*, *children*) → *owlapy.class\_expression.OWLQuantifiedObjectRestriction*

**visit\_cardinality\_res** (*node*, *children*) → *owlapy.class\_expression.OWLObjectCardinalityRestriction*

**visit\_value\_res** (*node*, *children*) → *owlapy.class\_expression.OWLObjectHasValue*

**visit\_has\_self** (*node*, *children*) → *owlapy.class\_expression.OWLObjectHasSelf*

**visit\_object\_property** (*node*, *children*) → *owlapy.owl\_property.OWLObjectPropertyExpression*

**visit\_class\_expression** (*node*, *children*) → *owlapy.class\_expression.OWLClassExpression*

**visit\_individual\_list** (*node*, *children*) → *owlapy.class\_expression.OWLObjectOneOf*

**visit\_data\_primary** (*node*, *children*) → *owlapy.owl\_data\_ranges.OWLDataRange*

**visit\_data\_some\_only\_res** (*node*, *children*)  
→ *owlapy.class\_expression.OWLQuantifiedDataRestriction*

**visit\_data\_cardinality\_res** (*node*, *children*)  
→ *owlapy.class\_expression.OWLDataCardinalityRestriction*

**visit\_data\_value\_res** (*node*, *children*) → *owlapy.class\_expression.OWLDataHasValue*

**visit\_data\_union** (*node*, *children*) → *owlapy.owl\_data\_ranges.OWLDataRange*

**visit\_data\_intersection** (*node*, *children*) → *owlapy.owl\_data\_ranges.OWLDataRange*

**visit\_literal\_list** (*node*, *children*) → *owlapy.class\_expression.OWLDataOneOf*

**visit\_data\_parentheses** (*node*, *children*) → *owlapy.owl\_data\_ranges.OWLDataRange*

**visit\_datatype\_restriction** (*node*, *children*) → *owlapy.class\_expression.OWLDatatypeRestriction*

**visit\_facet\_restrictions** (*node*, *children*) → List[*owlapy.class\_expression.OWLFacetRestriction*]

**visit\_facet\_restriction** (*node*, *children*) → *owlapy.class\_expression.OWLFacetRestriction*

**visit\_literal** (*node*, *children*) → *owlapy.owl\_literal.OWLLiteral*

**visit\_typed\_literal** (*node*, *children*) → *owlapy.owl\_literal.OWLLiteral*

**abstract\_visit\_string\_literal\_language** (*node*, *children*)

```

visit_string_literal_no_language (node, children) → owlapy.owl_literal.OWLLiteral
visit_quoted_string (node, children) → str
visit_float_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_decimal_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_integer_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_boolean_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_datetime_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_duration_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_date_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_non_negative_integer (node, children) → int
visit_datatype_iri (node, children) → str
visit_datatype (node, children) → owlapy.owl_datatype.OWLDatatype
visit_facet (node, children) → owlapy.vocab.OWLFacet
visit_class_iri (node, children) → owlapy.class_expression.OWLClass
visit_individual_iri (node, children) → owlapy.owl_individual.OWLNamedIndividual
visit_object_property_iri (node, children) → owlapy.owl_property.OWLObjectProperty
visit_data_property_iri (node, children) → owlapy.owl_property.OWLDataProperty
visit_iri (node, children) → owlapy.iri.IRI
visit_full_iri (node, children) → owlapy.iri.IRI
abstract visit_abbreviated_iri (node, children)
visit_simple_iri (node, children) → owlapy.iri.IRI
visit_parentheses (node, children) → owlapy.class_expression.OWLClassExpression
generic_visit (node, children)

```

Default visitor method

#### Parameters

- **node** – The node we’re visiting
- **visited\_children** – The results of visiting the children of that node, in a list

I’m not sure there’s an implementation of this that makes sense across all (or even most) use cases, so we leave it to subclasses to implement for now.

owlapy.parser.DL\_GRAMMAR

```

class owlapy.parser.DLSyntaxParser (namespace: str | owlapy.namespaces.Namespaces | None = None,
                                     grammar=None)

```

Bases: parsimonious.nodes.NodeVisitor, owlapy.owl\_object.OWLObjectParser

Description Logic Syntax parser to parse strings to OWLClassExpressions.

```

slots = ('ns', 'grammar')

ns: str | owlapy.namespaces.Namespaces | None

grammar

parse_expression(expression_str: str) → owlapy.class_expression.OWLClassExpression
    Parse a string to an OWL Object.

    Parameters
        expression_str (str) – Expression string.

    Returns
        The OWL Object which is represented by the string.

visit_union(node, children) → owlapy.class_expression.OWLClassExpression

visit_intersection(node, children) → owlapy.class_expression.OWLClassExpression

visit_primary(node, children) → owlapy.class_expression.OWLClassExpression

visit_some_only_res(node, children) → owlapy.class_expression.OWLQuantifiedObjectRestriction

visit_cardinality_res(node, children) → owlapy.class_expression.OWLObjectCardinalityRestriction

visit_value_res(node, children) → owlapy.class_expression.OWLObjectHasValue

visit_has_self(node, children) → owlapy.class_expression.OWLObjectHasSelf

visit_object_property(node, children) → owlapy.owl_property.OWLObjectPropertyExpression

visit_class_expression(node, children) → owlapy.class_expression.OWLClassExpression

visit_individual_list(node, children) → owlapy.class_expression.OWLObjectOneOf

visit_data_primary(node, children) → owlapy.owl_data_ranges.OWLDataRange

visit_data_some_only_res(node, children)
    → owlapy.class_expression.OWLQuantifiedDataRestriction

visit_data_cardinality_res(node, children)
    → owlapy.class_expression.OWLDataCardinalityRestriction

visit_data_value_res(node, children) → owlapy.class_expression.OWLDataHasValue

visit_data_union(node, children) → owlapy.owl_data_ranges.OWLDataRange

visit_data_intersection(node, children) → owlapy.owl_data_ranges.OWLDataRange

visit_literal_list(node, children) → owlapy.class_expression.OWLDataOneOf

visit_data_parentheses(node, children) → owlapy.owl_data_ranges.OWLDataRange

visit_datatype_restriction(node, children) → owlapy.class_expression.OWLDatatypeRestriction

visit_facet_restrictions(node, children) → List[owlapy.class_expression.OWLFacetRestriction]

visit_facet_restriction(node, children) → owlapy.class_expression.OWLFacetRestriction

visit_literal(node, children) → owlapy.owl_literal.OWLLiteral

```

```

visit_typed_literal (node, children) → owlapy.owl_literal.OWLLiteral
abstract visit_string_literal_language (node, children)
visit_string_literal_no_language (node, children) → owlapy.owl_literal.OWLLiteral
visit_quoted_string (node, children) → str
visit_float_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_decimal_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_integer_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_boolean_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_datetime_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_duration_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_date_literal (node, children) → owlapy.owl_literal.OWLLiteral
visit_non_negative_integer (node, children) → int
visit_datatype_iri (node, children) → str
visit_datatype (node, children) → owlapy.owl_datatype.OWLDatatype
visit_facet (node, children) → owlapy.vocab.OWLFacet
visit_class_iri (node, children) → owlapy.class_expression.OWLClass
visit_individual_iri (node, children) → owlapy.owl_individual.OWLNamedIndividual
visit_object_property_iri (node, children) → owlapy.owl_property.OWLObjectProperty
visit_data_property_iri (node, children) → owlapy.owl_property.OWLDataProperty
visit_iri (node, children) → owlapy.iri.IRI
visit_full_iri (node, children) → owlapy.iri.IRI
abstract visit_abbreviated_iri (node, children)
visit_simple_iri (node, children) → owlapy.iri.IRI
visit_parentheses (node, children) → owlapy.class_expression.OWLClassExpression
generic_visit (node, children)

```

Default visitor method

#### Parameters

- **node** – The node we’re visiting
- **visited\_children** – The results of visiting the children of that node, in a list

I’m not sure there’s an implementation of this that makes sense across all (or even most) use cases, so we leave it to subclasses to implement for now.

```
owlapy.parser.DLparser
```

owlapy.parser.**ManchesterParser**

owlapy.parser.**dl\_to\_owl\_expression** (*dl\_expression: str, namespace: str*)

owlapy.parser.**manchester\_to\_owl\_expression** (*manchester\_expression: str, namespace: str*)

## owlapy.providers

OWL Datatype restriction constructors.

## Attributes

---

*Restriction\_Literals*

---

## Functions

<i>owl_datatype_max_exclusive_restriction(...)</i>	Create a max exclusive restriction.
<i>owl_datatype_min_exclusive_restriction(...)</i>	Create a min exclusive restriction.
<i>owl_datatype_max_inclusive_restriction(...)</i>	Create a max inclusive restriction.
<i>owl_datatype_min_inclusive_restriction(...)</i>	Create a min inclusive restriction.
<i>owl_datatype_min_max_exclusive_restriction(...)</i>	Create a min-max exclusive restriction.
<i>owl_datatype_min_max_inclusive_restriction(...)</i>	Create a min-max inclusive restriction.

---

## Module Contents

owlapy.providers.**Restriction\_Literals**

owlapy.providers.**owl\_datatype\_max\_exclusive\_restriction** (*max\_: Restriction\_Literals*)  
→ *owlapy.class\_expression.OWLDatatypeRestriction*  
Create a max exclusive restriction.

owlapy.providers.**owl\_datatype\_min\_exclusive\_restriction** (*min\_: Restriction\_Literals*)  
→ *owlapy.class\_expression.OWLDatatypeRestriction*  
Create a min exclusive restriction.

owlapy.providers.**owl\_datatype\_max\_inclusive\_restriction** (*max\_: Restriction\_Literals*)  
→ *owlapy.class\_expression.OWLDatatypeRestriction*  
Create a max inclusive restriction.

owlapy.providers.**owl\_datatype\_min\_inclusive\_restriction** (*min\_: Restriction\_Literals*)  
→ *owlapy.class\_expression.OWLDatatypeRestriction*  
Create a min inclusive restriction.

owlapy.providers.**owl\_datatype\_min\_max\_exclusive\_restriction** (*min\_: Restriction\_Literals,*  
*max\_: Restriction\_Literals*) → *owlapy.class\_expression.OWLDatatypeRestriction*  
Create a min-max exclusive restriction.

owlapy.providers.**owl\_datatype\_min\_max\_inclusive\_restriction** (*min\_: Restriction\_Literals,*  
*max\_: Restriction\_Literals*) → *owlapy.class\_expression.OWLDatatypeRestriction*  
Create a min-max inclusive restriction.

## owlapy.render

Renderers for different syntax.

### Attributes

<i>mapper</i>
<i>DLrenderer</i>
<i>ManchesterRenderer</i>

### Classes

<i>DLSyntaxObjectRenderer</i>	DL Syntax renderer for OWL Objects.
<i>ManchesterOWLSyntaxOWLObjectRenderer</i>	Manchester Syntax renderer for OWL Objects

### Functions

<i>translating_short_form_provider</i> ( $\rightarrow$ str)	e: entity.
<i>translating_short_form_endpoint</i> ( $\rightarrow$ str)	Translates an OWLEntity to a short form string using provided rules and an endpoint.
<i>owl_expression_to_dl</i> ( $\rightarrow$ str)	
<i>owl_expression_to_manchester</i> ( $\rightarrow$ str)	

### Module Contents

owlapy.render.**mapper**

owlapy.render.**translating\_short\_form\_provider** (*e: owlapy.owl\_object.OWLEntity, reasoner, rules: dict[str:str] = None*)  $\rightarrow$  str

e: entity. reasoner: OWLReasoner or Triplestore(from Ontolearn) rules: A mapping from OWLEntity to predicates,

Keys in rules can be general or specific iris, e.g., IRI to IRI s.t. the second IRI must be a predicate leading to literal

owlapy.render.**translating\_short\_form\_endpoint** (*e: owlapy.owl\_object.OWLEntity, endpoint: str, rules: dict[abc.ABCMeta:str] = None*)  $\rightarrow$  str

Translates an OWLEntity to a short form string using provided rules and an endpoint.

Parameters: e (OWLEntity): The OWL entity to be translated. endpoint (str): The endpoint of a triple store to query against. rules (dict[abc.ABCMeta:str], optional): A dictionary mapping OWL classes to string IRIs leading to a literal.

Returns: str: The translated short form of the OWL entity. If no matching rules are found, a simple short form is returned.

This function iterates over the provided rules to check if the given OWL entity is an instance of any specified class. If a match is found, it constructs a SPARQL query to retrieve the literal value associated with the entity and predicate. If a literal is found, it is returned as the short form. If no literals are found, the SPARQL query and entity information are printed for debugging purposes. If no matching rules are found, a warning is issued and a simple short form is returned.

Example: >>> e = OWLEntity("http://example.org/entity") >>> endpoint = "http://example.org/sparql" >>> rules = {SomeOWLClass: "http://example.org/predicate"} >>> translating\_short\_form\_endpoint(e, endpoint, rules)

```
class owlapy.render.DLSyntaxObjectRenderer (
    short_form_provider: Callable[[owlapy.owl_object.OWLEntity], str] = _simple_short_form_provider)
```

Bases: *owlapy.owl\_object.OWLObjectRenderer*

DL Syntax renderer for OWL Objects.

**\_\_slots\_\_** = `'_sfp'`

**set\_short\_form\_provider** (*short\_form\_provider: Callable[[owlapy.owl\_object.OWLEntity], str]*)  
→ None

Configure a short form provider that shortens the OWL objects during rendering.

#### Parameters

**short\_form\_provider** – Short form provider.

**render** (*o: owlapy.owl\_object.OWLObject*) → str

Render OWL Object to string.

#### Parameters

**o** – OWL Object.

#### Returns

String rendition of OWL object.

```
class owlapy.render.ManchesterOWLSyntaxOWLObjectRenderer (
    short_form_provider: Callable[[owlapy.owl_object.OWLEntity], str] = _simple_short_form_provider,
    no_render_thing=False)
```

Bases: *owlapy.owl\_object.OWLObjectRenderer*

Manchester Syntax renderer for OWL Objects

**\_\_slots\_\_** = `('_sfp', '_no_render_thing')`

**set\_short\_form\_provider** (*short\_form\_provider: Callable[[owlapy.owl\_object.OWLEntity], str]*)  
→ None

Configure a short form provider that shortens the OWL objects during rendering.

#### Parameters

**short\_form\_provider** – Short form provider.

**render** (*o: owlapy.owl\_object.OWLObject*) → str

Render OWL Object to string.

#### Parameters

**o** – OWL Object.

#### Returns

String rendition of OWL object.

`owlapy.render.DLrenderer`

`owlapy.render.ManchesterRenderer`

`owlapy.render.owl_expression_to_dl(o: owlapy.owl_object.OWLObject) → str`

`owlapy.render.owl_expression_to_manchester(o: owlapy.owl_object.OWLObject) → str`

## owlapy.static\_funcs

Static functions for general purposes.

### Functions

<code>move(*args)</code>	"Move" an imported class to the current module by setting the classes <code>__module__</code> attribute.
<code>download_external_files(ftp_link)</code>	
<code>startJVM()</code>	Start the JVM with jar dependencies. This method is called automatically on object initialization, if the
<code>stopJVM(→ None)</code>	Detaches the thread from Java packages and shuts down the java virtual machine hosted by jpye.
<code>create_ontology(iri[, with_owlapi])</code>	A convenient function

### Module Contents

`owlapy.static_funcs.move(*args)`

"Move" an imported class to the current module by setting the classes `__module__` attribute.

This is useful for documentation purposes to hide internal packages in sphinx.

#### Parameters

**args** – List of classes to move.

`owlapy.static_funcs.download_external_files(ftp_link: str)`

`owlapy.static_funcs.startJVM()`

Start the JVM with jar dependencies. This method is called automatically on object initialization, if the JVM is not started yet.

`owlapy.static_funcs.stopJVM() → None`

Detaches the thread from Java packages and shuts down the java virtual machine hosted by jpye.

`owlapy.static_funcs.create_ontology(iri, with_owlapi=False)`

A convenient function

## owlapy.util\_owl\_static\_funcs

### Functions

<code>save_owl_class_expressions(→ None)</code>	Saves a set of OWL class expressions to an ontology file in RDF/XML format.
<code>csv_to_rdf_kg([path_csv, path_kg, namespace])</code>	Transforms a CSV file to an RDF Knowledge Graph in RDF/XML format.



## Module Contents

`owlapy.util_owl_static_funcs.save_owl_class_expressions (`  
    `expressions: owlapy.class_expression.OWLClassExpression | List[owlapy.class_expression.OWLClassExpression],`  
    `path: str = 'predictions', rdf_format: str = 'rdxml', namespace: str = None) → None`

Saves a set of OWL class expressions to an ontology file in RDF/XML format.

This function takes one or more OWL class expressions, creates an ontology, and saves the expressions as OWL equivalent class axioms in the specified RDF format. By default, it saves the file to the specified path using the 'rdxml' format.

### Parameters

- **expressions** (`OWLClassExpression | List[OWLClassExpression]`) – A single or a list of OWL class expressions to be saved as equivalent class axioms.
- **path** (`str, optional`) – The file path where the ontology will be saved. Defaults to 'predictions'.
- **rdxml\_format** (`str, optional`) – RDF serialization format for saving the ontology. Currently only supports 'rdxml'. Defaults to 'rdxml'.
- **namespace** (`str, optional`) – The namespace URI used for the ontology. If None, defaults to 'https://dice-research.org/predictions#'. Must end with '#'.

### Raises

- **AssertionError** – If *expressions* is neither an `OWLClassExpression` nor a list of `OWLClassExpression`.
- **AssertionError** – If *rdxml\_format* is not 'rdxml'.
- **AssertionError** – If *namespace* does not end with a '#'.

### Example

```
>>> from some_module import OWLClassExpression
>>> expr1 = OWLClassExpression("SomeExpression1")
>>> expr2 = OWLClassExpression("SomeExpression2")
>>> save_owl_class_expressions([expr1, expr2], path="my_ontology.owl", rdf_format=
↪ "rdxml")
```

`owlapy.util_owl_static_funcs.csv_to_rdf_kg (path_csv: str = None, path_kg: str = None,`  
    `namespace: str = None)`

Transforms a CSV file to an RDF Knowledge Graph in RDF/XML format.

### Parameters

- **path\_csv** (`str`) – X
- **path\_kg** (`str`) – X
- **namespace** (`str`) – X

### Raises

**AssertionError** –

## Example

```
>>> from sklearn.datasets import load_iris
>>> import pandas as pd
# Load the dataset
>>> data = load_iris()
# Convert to DataFrame
>>> df = pd.DataFrame(data.data, columns=data.feature_names)
>>> df['target'] = data.target
# Save as CSV
>>> df.to_csv("iris_dataset.csv", index=False)
>>> print("Dataset saved as iris_dataset.csv")
>>> csv_to_rdf_kg("iris_dataset.csv")
```

## owlapy.utils

Owlapy utils.

## Attributes

*measurer*

## Classes

<i>OWLClassExpressionLengthMetric</i>	Length calculation of OWLClassExpression
<i>EvaluatedDescriptionSet</i>	Abstract base class for generic types.
<i>ConceptOperandSorter</i>	
<i>OperandSetTransform</i>	
<i>HasIndex</i>	Interface for types with an index; this is used to group objects by type when sorting.
<i>OrderedOWLObject</i>	Holder of OWL Objects that can be used for Python sorted.
<i>NNF</i>	This class contains functions to transform a Class Expression into Negation Normal Form.
<i>TopLevelCNF</i>	This class contains functions to transform a class expression into Top-Level Conjunctive Normal Form.
<i>TopLevelDNF</i>	This class contains functions to transform a class expression into Top-Level Disjunctive Normal Form.
<i>LRUCache</i>	Constants shares by all lru cache instances.

## Functions

<code>run_with_timeout(func, timeout[, args])</code>	
<code>concept_reducer(concepts, opt)</code>	Reduces a set of concepts by applying a binary operation to each pair of concepts.
<code>concept_reducer_properties(...)</code>	Map a set of owl concepts and a set of properties into OWL Restrictions
<code>get_expression_length(→ int)</code>	
<code>combine_nary_expressions(...)</code>	Shortens an OWLClassExpression or OWLDataRange by combining all nested nary expressions of the same type.
<code>iter_count(→ int)</code>	Count the number of elements in an iterable.
<code>as_index(→ HasIndex)</code>	Cast OWL Object to HasIndex.

## Module Contents

`owlapy.utils.run_with_timeout(func, timeout, args=(), **kwargs)`

`owlapy.utils.concept_reducer(concepts: Iterable, opt: Callable)`

Reduces a set of concepts by applying a binary operation to each pair of concepts.

### Parameters

- **concepts** (*set*) – A set of concepts to be reduced.
- **opt** (*function*) – A binary function that takes a pair of concepts and returns a single concept.

### Returns

A set containing the results of applying the binary operation to each pair of concepts.

### Return type

set

## Example

```
>>> concepts = {1, 2, 3}
>>> opt = lambda x: x[0] + x[1]
>>> concept_reducer(concepts, opt)
{2, 3, 4, 5, 6}
```

### Note

The operation *opt* should be commutative and associative to ensure meaningful reduction in the context of set operations.

`owlapy.utils.concept_reducer_properties(concepts: Iterable, properties, cls: Callable = None, cardinality: int = 2)`

→ `Iterable[owlapy.class_expression.OWLQuantifiedObjectRestriction | owlapy.class_expression.OWLObjectCardinalityRestriction]`

Map a set of owl concepts and a set of properties into OWL Restrictions

### Parameters

- **concepts**

- **properties**
- **cls** (*Callable*) – An owl Restriction class
- **cardinality** – A positive Integer

Returns: List of OWL Restrictions

```
class owlapy.utils.OWLClassExpressionLengthMetric(*, class_length: int,
    object_intersection_length: int, object_union_length: int, object_complement_length: int,
    object_some_values_length: int, object_all_values_length: int, object_has_value_length: int,
    object_cardinality_length: int, object_has_self_length: int, object_one_of_length: int,
    data_some_values_length: int, data_all_values_length: int, data_has_value_length: int,
    data_cardinality_length: int, object_property_length: int, object_inverse_length: int,
    data_property_length: int, datatype_length: int, data_one_of_length: int,
    data_complement_length: int, data_intersection_length: int, data_union_length: int)
```

Length calculation of OWLClassExpression

#### Parameters

- **class\_length** – Class: “C”
- **object\_intersection\_length** – Intersection:  $A \sqcap B$
- **object\_union\_length** – Union:  $A \sqcup B$
- **object\_complement\_length** – Complement:  $\neg C$
- **object\_some\_values\_length** – Obj. Some Values:  $\exists r.C$
- **object\_all\_values\_length** – Obj. All Values:  $\forall r.C$
- **object\_has\_value\_length** – Obj. Has Value:  $\exists r.\{I\}$
- **object\_cardinality\_length** – Obj. Cardinality restriction:  $\leq n r.C$
- **object\_has\_self\_length** – Obj. Self restriction:  $\exists r.\text{Self}$
- **object\_one\_of\_length** – Obj. One of:  $\exists r.\{X,Y,Z\}$
- **data\_some\_values\_length** – Data Some Values:  $\exists p.t$
- **data\_all\_values\_length** – Data All Values:  $\forall p.t$
- **data\_has\_value\_length** – Data Has Value:  $\exists p.\{V\}$
- **data\_cardinality\_length** – Data Cardinality restriction:  $\leq n r.t$
- **object\_property\_length** – Obj. Property:  $\exists r.C$
- **object\_inverse\_length** – Inverse property:  $\exists r^{-}.C$
- **data\_property\_length** – Data Property:  $\exists p.t$
- **datatype\_length** – Datatype:  $^{\wedge}\text{datatype}$
- **data\_one\_of\_length** – Data One of:  $\exists p.\{U,V,W\}$
- **data\_complement\_length** – Data Complement:  $\neg\text{datatype}$
- **data\_intersection\_length** – Data Intersection:  $\text{datatype} \sqcap \text{datatype}$
- **data\_union\_length** – Data Union:  $\text{datatype} \sqcup \text{datatype}$

```
__slots__ = ('class_length', 'object_intersection_length',
    'object_union_length', ...)
```

```

class_length: int
object_intersection_length: int
object_union_length: int
object_complement_length: int
object_some_values_length: int
object_all_values_length: int
object_has_value_length: int
object_cardinality_length: int
object_has_self_length: int
object_one_of_length: int
data_some_values_length: int
data_all_values_length: int
data_has_value_length: int
data_cardinality_length: int
object_property_length: int
object_inverse_length: int
data_property_length: int
datatype_length: int
data_one_of_length: int
data_complement_length: int
data_intersection_length: int
data_union_length: int

static get_default () → OWLClassExpressionLengthMetric

abstract length(o: owlapy.owl_object.OWLObject) → int

```

owlapy.utils.measurer

```

owlapy.utils.get_expression_length(ce: owlapy.class_expression.OWLClassExpression) → int

```

**class** owlapy.utils.EvaluatedDescriptionSet (ordering: Callable[[\_N], \_O], max\_size: int = 10)

Bases: Generic[\_N, \_O]

Abstract base class for generic types.

A generic type is typically declared by inheriting from this class parameterized with one or more type variables. For example, a generic mapping type might be defined as:

```
class Mapping(Generic[KT, VT]):
    def __getitem__(self, key: KT) -> VT:
        ...
    # Etc.
```

This class can then be used as follows:

```
def lookup_name(mapping: Mapping[KT, VT], key: KT, default: VT) -> VT:
    try:
        return mapping[key]
    except KeyError:
        return default
```

```
__slots__ = ('items', '_max_size', '_Ordering')
```

```
items: SortedSet[_N]
```

```
maybe_add(node: _N)
```

```
clean()
```

```
worst()
```

```
best()
```

```
best_quality_value() -> float
```

```
__iter__() -> Iterable[_N]
```

```
class owlapy.utils.ConceptOperandSorter
```

```
    abstract sort(o: _O) -> _O
```

```
class owlapy.utils.OperandSetTransform
```

```
    simplify(o: owlapy.class_expression.OWLClassExpression)
        -> owlapy.class_expression.OWLClassExpression
```

```
class owlapy.utils.HasIndex
```

```
    Bases: Protocol
```

Interface for types with an index; this is used to group objects by type when sorting.

```
    type_index: ClassVar[int]
```

```
    __eq__(other)
```

```
class owlapy.utils.OrderedOWLObject(o: _HasIndex)
```

Holder of OWL Objects that can be used for Python sorted.

The Ordering is dependent on the type\_index of the impl. classes recursively followed by all components of the OWL Object.

o

OWL object.

```
__slots__ = ('o', '_chain')
```

`o: _HasIndex`

`__lt__(other)`

`__eq__(other)`

`class owlapy.utils.NNF`

This class contains functions to transform a Class Expression into Negation Normal Form.

**abstract** `get_class_nnf` (*ce: owlapy.class\_expression.OWLClassExpression, negated: bool = False*)  
→ *owlapy.class\_expression.OWLClassExpression*

Convert a Class Expression to Negation Normal Form. Operands will be sorted.

**Parameters**

- **ce** – Class Expression.
- **negated** – Whether the result should be negated.

**Returns**

Class Expression in Negation Normal Form.

`class owlapy.utils.TopLevelCNF`

This class contains functions to transform a class expression into Top-Level Conjunctive Normal Form.

**get\_top\_level\_cnf** (*ce: owlapy.class\_expression.OWLClassExpression*)  
→ *owlapy.class\_expression.OWLClassExpression*

Convert a class expression into Top-Level Conjunctive Normal Form. Operands will be sorted.

**Parameters**

**ce** – Class Expression.

**Returns**

Class Expression in Top-Level Conjunctive Normal Form.

`class owlapy.utils.TopLevelDNF`

This class contains functions to transform a class expression into Top-Level Disjunctive Normal Form.

**get\_top\_level\_dnf** (*ce: owlapy.class\_expression.OWLClassExpression*)  
→ *owlapy.class\_expression.OWLClassExpression*

Convert a class expression into Top-Level Disjunctive Normal Form. Operands will be sorted.

**Parameters**

**ce** – Class Expression.

**Returns**

Class Expression in Top-Level Disjunctive Normal Form.

`owlapy.utils.combine_nary_expressions` (*ce: owlapy.class\_expression.OWLClassExpression*)  
→ *owlapy.class\_expression.OWLClassExpression*

`owlapy.utils.combine_nary_expressions` (*ce: owlapy.owl\_data\_ranges.OWLDataRange*)  
→ *owlapy.owl\_data\_ranges.OWLDataRange*

Shortens an OWLClassExpression or OWLDataRange by combining all nested nary expressions of the same type. Operands will be sorted.

E.g. OWLObjectUnionOf(A, OWLObjectUnionOf(C, B)) -> OWLObjectUnionOf(A, B, C).

`owlapy.utils.iter_count` (*i: Iterable*) → int

Count the number of elements in an iterable.

`owlapy.utils.as_index(o: owlapy.owl_object.OWLObject) → HasIndex`

Cast OWL Object to HasIndex.

**class** `owlapy.utils.LRUCache` (*maxsize: int | None = None*)

**Bases:** `Generic[_K, _V]`

Constants shares by all lru cache instances.

Adapted from `functools.lru_cache`.

**sentinel**

Unique object used to signal cache misses.

**PREV**

Name for the link field 0.

**NEXT**

Name for the link field 1.

**KEY**

Name for the link field 2.

**RESULT**

Name for the link field 3.

**sentinel**

**cache**

**full = False**

**cache\_get**

**cache\_len**

**lock**

**root = []**

**maxsize**

**\_\_contains\_\_** (*item: \_K*) → bool

**\_\_getitem\_\_** (*item: \_K*) → \_V

**\_\_setitem\_\_** (*key: \_K, value: \_V*)

**cache\_info** ()

Report cache statistics.

**cache\_clear** ()

Clear the cache and cache statistics.

## **owlapy.vocab**

Enumerations.



## Classes

<i>OWLRDFVocabulary</i>	Enumerations for OWL/RDF vocabulary.
<i>XSDVocabulary</i>	Enumerations for XSD vocabulary.
<i>OWLFacet</i>	Enumerations for OWL facets.

## Module Contents

**class** owlapy.vocab.OWLRDFVocabulary (namespace: *owlapy.namespaces.Namespaces*, remainder: *str*)

Bases: *\_Vocabulary*, *enum.Enum*

Enumerations for OWL/RDF vocabulary.

OWL\_THING

OWL\_NOTHING

OWL\_CLASS

OWL\_NAMED\_INDIVIDUAL

OWL\_TOP\_OBJECT\_PROPERTY

OWL\_BOTTOM\_OBJECT\_PROPERTY

OWL\_TOP\_DATA\_PROPERTY

OWL\_BOTTOM\_DATA\_PROPERTY

RDFS\_LITERAL

**class** owlapy.vocab.XSDVocabulary (remainder: *str*)

Bases: *\_Vocabulary*, *enum.Enum*

Enumerations for XSD vocabulary.

DECIMAL: Final = 'decimal'

INTEGER: Final = 'integer'

NONNEGATIVEINTEGER: Final = 'nonNegativeInteger'

NONPOSITIVEINTEGER: Final = 'nonPositiveInteger'

POSITIVEINTEGER: Final = 'positiveInteger'

NEGATIVEINTEGER: Final = 'negativeInteger'

LONG: Final = 'long'

DOUBLE: Final = 'double'

FLOAT: Final = 'float'

BOOLEAN: Final = 'boolean'

STRING: Final = 'string'

```

DATE: Final = 'date'

DATE_TIME: Final = 'dateTime'

DATE_TIME_STAMP: Final = 'dateTimeStamp'

DURATION: Final = 'duration'

TIME: Final = 'time'

GYEARMONTH: Final = 'gYearMonth'

GMONTHDAY: Final = 'gMonthDay'

GYEAR: Final = 'gYear'

GMONTH: Final = 'gMonth'

GDAY: Final = 'gDay'

class owlapy.vocab.OWLFacet (remainder: str, symbolic_form: str, operator: Callable[[_X, _X], bool])
    Bases: _Vocabulary, enum.Enum
    Enumerations for OWL facets.
    property symbolic_form
    property operator
    static from_str(name: str) → OWLFacet
    MIN_INCLUSIVE: Final
    MIN_EXCLUSIVE: Final
    MAX_INCLUSIVE: Final
    MAX_EXCLUSIVE: Final
    LENGTH: Final
    MIN_LENGTH: Final
    MAX_LENGTH: Final
    PATTERN: Final
    TOTAL_DIGITS: Final
    FRACTION_DIGITS: Final

```

## 7.2 Classes

*OntologyManager*

An OWLOntologyManager manages a set of ontologies.  
It is the main point for creating, loading and accessing

## 7.3 Functions

<code>owl_expression_to_dl(→ str)</code>	
<code>owl_expression_to_manchester(→ str)</code>	
<code>dl_to_owl_expression(dl_expression, namespace)</code>	
<code>manchester_to_owl_expression(manchester_expres ...)</code>	
<code>owl_expression_to_sparql(→ str)</code>	Convert an OWL Class Expression ( <a href="https://www.w3.org/TR/owl2-syntax/#Class_Expressions">https://www.w3.org/TR/owl2-syntax/#Class_Expressions</a> ) into a SPARQL query
<code>owl_expression_to_sparql_with_confusion_ma: str)</code>	Convert an OWL Class Expression ( <a href="https://www.w3.org/TR/owl2-syntax/#Class_Expressions">https://www.w3.org/TR/owl2-syntax/#Class_Expressions</a> ) into a SPARQL query

## 7.4 Package Contents

`owlapy.owl_expression_to_dl(o: owlapy.owl_object.OWLObject) → str`

`owlapy.owl_expression_to_manchester(o: owlapy.owl_object.OWLObject) → str`

`owlapy.dl_to_owl_expression(dl_expression: str, namespace: str)`

`owlapy.manchester_to_owl_expression(manchester_expression: str, namespace: str)`

`owlapy.owl_expression_to_sparql(expression: owlapy.class_expression.OWLClassExpression = None,  
root_variable: str = '?x',  
values: Iterable[owlapy.owl_individual.OWLNamedIndividual] | None = None,  
for_all_de_morgan: bool = True, named_individuals: bool = False) → str`

Convert an OWL Class Expression ([https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions)) into a SPARQL query  
root\_variable: the variable that will be projected expression: the class expression to be transformed to a SPARQL query

values: positive or negative examples from a class expression problem. Unclear for\_all\_de\_morgan: if set to True, the SPARQL mapping will use the mapping containing the nested FILTER NOT EXISTS patterns for the universal quantifier ( $\neg(\exists r. \neg C)$ ), instead of the counting query named\_individuals: if set to True, the generated SPARQL query will return only entities that are instances of owl:NamedIndividual

`owlapy.owl_expression_to_sparql_with_confusion_matrix(  
expression: owlapy.class_expression.OWLClassExpression,  
positive_examples: Iterable[owlapy.owl_individual.OWLNamedIndividual] | None,  
negative_examples: Iterable[owlapy.owl_individual.OWLNamedIndividual] | None,  
root_variable: str = '?x', for_all_de_morgan: bool = True, named_individuals: bool = False) → str`

Convert an OWL Class Expression ([https://www.w3.org/TR/owl2-syntax/#Class\\_Expressions](https://www.w3.org/TR/owl2-syntax/#Class_Expressions)) into a SPARQL query  
root\_variable: the variable that will be projected expression: the class expression to be transformed to a SPARQL query  
positive\_examples: positive examples from a class expression problem  
negative\_examples: negative examples from a class expression problem  
for\_all\_de\_morgan: if set to True, the SPARQL mapping will use the mapping containing the nested FILTER NOT EXISTS patterns for the universal quantifier ( $\neg(\exists r. \neg C)$ ), instead of the counting query named\_individuals: if set to True, the generated SPARQL query will return only entities that are instances of owl:NamedIndividual

**class** `owlapy.OntologyManager(world_store=None)`

Bases: `owlapy.abstracts.abstract_owl_ontology_manager.AbstractOWLontologyManager`

An OWLOntologyManager manages a set of ontologies. It is the main point for creating, loading and accessing ontologies.

`__slots__ = '_world'`

**create\_ontology** (*iri: str | owlapy.iri.IRI = None*) → *owlapy.owl\_ontology.Ontology*

Creates a new (empty) ontology that has the specified ontology IRI (and no version IRI).

**Parameters**

**iri** – The IRI of the ontology to be created, can also be a string.

**Returns**

The newly created ontology.

**load\_ontology** (*path: str = None*) → *owlapy.owl\_ontology.Ontology*

Loads an ontology that is assumed to have the specified ontology IRI as its IRI or version IRI. The ontology IRI will be mapped to an ontology document IRI.

**Parameters**

**iri** –

**The IRI that identifies the ontology, can also be a string.**

It is expected that the ontology will also have this IRI

(although the OWL API should tolerate situations where this is not the case).

**Returns**

The OWLOntology representation of the ontology that was loaded.

**apply\_change** (*change: owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyChange*)

A convenience method that applies just one change to an ontology. When this method is used through an OWLOntologyManager implementation, the instance used should be the one that the ontology returns through the `get_owl_ontology_manager()` call.

**Parameters**

**change** – The change to be applied.

**Raises**

**ChangeApplied.UNSUCCESSFULLY** – if the change was not applied successfully.

**save\_world()**

Saves the actual state of the quadstore in the SQLite3 file.

## Python Module Index

### O

- owlapy, 15
- owlapy.abstracts, 15
- owlapy.abstracts.abstract\_owl\_ontology, 15
- owlapy.abstracts.abstract\_owl\_ontology\_manager, 18
- owlapy.abstracts.abstract\_owl\_reasoner, 19
- owlapy.class\_expression, 36
- owlapy.class\_expression.class\_expression, 36
- owlapy.class\_expression.nary\_boolean\_expression, 38
- owlapy.class\_expression.owl\_class, 39
- owlapy.class\_expression.restriction, 40
- owlapy.converter, 67
- owlapy.entities, 70
- owlapy.iri, 70
- owlapy.meta\_classes, 72
- owlapy.namespaces, 73
- owlapy.owl\_annotation, 74
- owlapy.owl\_axiom, 75
- owlapy.owl\_data\_ranges, 94
- owlapy.owl\_datatype, 95
- owlapy.owl\_hierarchy, 96
- owlapy.owl\_individual, 100
- owlapy.owl\_literal, 100
- owlapy.owl\_object, 107
- owlapy.owl\_ontology, 108
- owlapy.owl\_ontology\_manager, 116
- owlapy.owl\_property, 118
- owlapy.owl\_reasoner, 122
- owlapy.owlapi\_mapper, 135
- owlapy.parser, 136
- owlapy.providers, 141
- owlapy.render, 142
- owlapy.static\_funcs, 144
- owlapy.util\_owl\_static\_funcs, 144
- owlapy.utils, 146
- owlapy.vocab, 152

## Index

### Non-alphabetical

`__contains__()` (*owlapy.converter.VariablesMapping method*), 68  
`__contains__()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 98  
`__contains__()` (*owlapy.utils.LRUCache method*), 152  
`__eq__()` (*owlapy.class\_expression.class\_expression.OWLObjectComplementOf method*), 38  
`__eq__()` (*owlapy.class\_expression.nary\_boolean\_expression.OWLNaryBooleanClassExpression method*), 38  
`__eq__()` (*owlapy.class\_expression.OWLDataAllValuesFrom method*), 65  
`__eq__()` (*owlapy.class\_expression.OWLDataCardinalityRestriction method*), 61  
`__eq__()` (*owlapy.class\_expression.OWLDataHasValue method*), 66  
`__eq__()` (*owlapy.class\_expression.OWLDataOneOf method*), 60  
`__eq__()` (*owlapy.class\_expression.OWLDataSomeValuesFrom method*), 65  
`__eq__()` (*owlapy.class\_expression.OWLDatatypeRestriction method*), 63  
`__eq__()` (*owlapy.class\_expression.OWLFacetRestriction method*), 63  
`__eq__()` (*owlapy.class\_expression.OWLHasValueRestriction method*), 58  
`__eq__()` (*owlapy.class\_expression.OWLNaryBooleanClassExpression method*), 57  
`__eq__()` (*owlapy.class\_expression.OWLObjectAllValuesFrom method*), 62  
`__eq__()` (*owlapy.class\_expression.OWLObjectCardinalityRestriction method*), 60  
`__eq__()` (*owlapy.class\_expression.OWLObjectComplementOf method*), 55  
`__eq__()` (*owlapy.class\_expression.OWLObjectHasSelf method*), 60  
`__eq__()` (*owlapy.class\_expression.OWLObjectOneOf method*), 67  
`__eq__()` (*owlapy.class\_expression.OWLObjectSomeValuesFrom method*), 61  
`__eq__()` (*owlapy.class\_expression.restriction.OWLDataAllValuesFrom method*), 50  
`__eq__()` (*owlapy.class\_expression.restriction.OWLDataCardinalityRestriction method*), 49  
`__eq__()` (*owlapy.class\_expression.restriction.OWLDataHasValue method*), 51  
`__eq__()` (*owlapy.class\_expression.restriction.OWLDataOneOf method*), 51  
`__eq__()` (*owlapy.class\_expression.restriction.OWLDataSomeValuesFrom method*), 50  
`__eq__()` (*owlapy.class\_expression.restriction.OWLDatatypeRestriction method*), 52  
`__eq__()` (*owlapy.class\_expression.restriction.OWLFacetRestriction method*), 52  
`__eq__()` (*owlapy.class\_expression.restriction.OWLHasValueRestriction method*), 43  
`__eq__()` (*owlapy.class\_expression.restriction.OWLObjectAllValuesFrom method*), 46  
`__eq__()` (*owlapy.class\_expression.restriction.OWLObjectCardinalityRestriction method*), 45  
`__eq__()` (*owlapy.class\_expression.restriction.OWLObjectHasSelf method*), 47  
`__eq__()` (*owlapy.class\_expression.restriction.OWLObjectOneOf method*), 48  
`__eq__()` (*owlapy.class\_expression.restriction.OWLObjectSomeValuesFrom method*), 46  
`__eq__()` (*owlapy.iri.IRI method*), 71  
`__eq__()` (*owlapy.namespaces.Namespaces method*), 74  
`__eq__()` (*owlapy.owl\_axiom.OWLAnnotation method*), 85  
`__eq__()` (*owlapy.owl\_axiom.OWLAnnotationAssertionAxiom method*), 86  
`__eq__()` (*owlapy.owl\_axiom.OWLAnnotationPropertyDomainAxiom method*), 87  
`__eq__()` (*owlapy.owl\_axiom.OWLAnnotationPropertyRangeAxiom method*), 87  
`__eq__()` (*owlapy.owl\_axiom.OWLClassAssertionAxiom method*), 84  
`__eq__()` (*owlapy.owl\_axiom.OWLDataPropertyCharacteristicAxiom method*), 92  
`__eq__()` (*owlapy.owl\_axiom.OWLDatatypeDefinitionAxiom method*), 79  
`__eq__()` (*owlapy.owl\_axiom.OWLDeclarationAxiom method*), 78  
`__eq__()` (*owlapy.owl\_axiom.OWLDisjointUnionAxiom method*), 84  
`__eq__()` (*owlapy.owl\_axiom.OWLHasKeyAxiom method*), 79  
`__eq__()` (*owlapy.owl\_axiom.OWLNaryClassAxiom method*), 80  
`__eq__()` (*owlapy.owl\_axiom.OWLNaryIndividualAxiom method*), 81  
`__eq__()` (*owlapy.owl\_axiom.OWLNaryPropertyAxiom method*), 82  
`__eq__()` (*owlapy.owl\_axiom.OWLObjectPropertyCharacteristicAxiom method*), 90  
`__eq__()` (*owlapy.owl\_axiom.OWLPropertyAssertionAxiom method*), 88  
`__eq__()` (*owlapy.owl\_axiom.OWLPropertyDomainAxiom method*), 92  
`__eq__()` (*owlapy.owl\_axiom.OWLPropertyRangeAxiom method*), 92  
`__eq__()` (*owlapy.owl\_axiom.OWLSubAnnotationPropertyOfAxiom method*), 86  
`__eq__()` (*owlapy.owl\_axiom.OWLSubClassOfAxiom method*), 84  
`__eq__()` (*owlapy.owl\_axiom.OWLSubPropertyAxiom method*), 87  
`__eq__()` (*owlapy.owl\_data\_ranges.OWLDataComplementOf method*), 95  
`__eq__()` (*owlapy.owl\_data\_ranges.OWLNaryDataRange method*), 94  
`__eq__()` (*owlapy.owl\_object.OWLNamedObject method*), 108  
`__eq__()` (*owlapy.owl\_object.OWLObject method*), 107  
`__eq__()` (*owlapy.owl\_ontology.Ontology method*), 112  
`__eq__()` (*owlapy.owl\_ontology.OWLOntologyID method*), 110  
`__eq__()` (*owlapy.owl\_ontology.SyncOntology method*), 113  
`__eq__()` (*owlapy.owl\_property.OWLObjectInverseOf method*), 121  
`__eq__()` (*owlapy.utils.HasIndex method*), 150

- `__eq__()` (*owlapy.utils.OrderedOWLObject method*), 151
- `__getitem__()` (*owlapy.converter.VariablesMapping method*), 68
- `__getitem__()` (*owlapy.utils.LRUCache method*), 152
- `__hash__()` (*owlapy.class\_expression.class\_expression.OWLObjectComplementOf method*), 38
- `__hash__()` (*owlapy.class\_expression.nary\_boolean\_expression.OWLNaryBooleanClassExpression method*), 38
- `__hash__()` (*owlapy.class\_expression.OWLDataAllValuesFrom method*), 65
- `__hash__()` (*owlapy.class\_expression.OWLDataCardinalityRestriction method*), 61
- `__hash__()` (*owlapy.class\_expression.OWLDataHasValue method*), 66
- `__hash__()` (*owlapy.class\_expression.OWLDataOneOf method*), 60
- `__hash__()` (*owlapy.class\_expression.OWLDataSomeValuesFrom method*), 65
- `__hash__()` (*owlapy.class\_expression.OWLDatatypeRestriction method*), 63
- `__hash__()` (*owlapy.class\_expression.OWLFacetRestriction method*), 63
- `__hash__()` (*owlapy.class\_expression.OWLHasValueRestriction method*), 58
- `__hash__()` (*owlapy.class\_expression.OWLNaryBooleanClassExpression method*), 57
- `__hash__()` (*owlapy.class\_expression.OWLObjectAllValuesFrom method*), 62
- `__hash__()` (*owlapy.class\_expression.OWLObjectCardinalityRestriction method*), 60
- `__hash__()` (*owlapy.class\_expression.OWLObjectComplementOf method*), 55
- `__hash__()` (*owlapy.class\_expression.OWLObjectHasSelf method*), 60
- `__hash__()` (*owlapy.class\_expression.OWLObjectOneOf method*), 67
- `__hash__()` (*owlapy.class\_expression.OWLObjectSomeValuesFrom method*), 61
- `__hash__()` (*owlapy.class\_expression.restriction.OWLDataAllValuesFrom method*), 50
- `__hash__()` (*owlapy.class\_expression.restriction.OWLDataCardinalityRestriction method*), 49
- `__hash__()` (*owlapy.class\_expression.restriction.OWLDataHasValue method*), 51
- `__hash__()` (*owlapy.class\_expression.restriction.OWLDataOneOf method*), 51
- `__hash__()` (*owlapy.class\_expression.restriction.OWLDataSomeValuesFrom method*), 50
- `__hash__()` (*owlapy.class\_expression.restriction.OWLDatatypeRestriction method*), 52
- `__hash__()` (*owlapy.class\_expression.restriction.OWLFacetRestriction method*), 52
- `__hash__()` (*owlapy.class\_expression.restriction.OWLHasValueRestriction method*), 43
- `__hash__()` (*owlapy.class\_expression.restriction.OWLObjectAllValuesFrom method*), 46
- `__hash__()` (*owlapy.class\_expression.restriction.OWLObjectCardinalityRestriction method*), 45
- `__hash__()` (*owlapy.class\_expression.restriction.OWLObjectHasSelf method*), 47
- `__hash__()` (*owlapy.class\_expression.restriction.OWLObjectOneOf method*), 48
- `__hash__()` (*owlapy.class\_expression.restriction.OWLObjectSomeValuesFrom method*), 46
- `__hash__()` (*owlapy.iri.IRI method*), 71
- `__hash__()` (*owlapy.namespaces.Namespaces method*), 74
- `__hash__()` (*owlapy.owl\_axiom.OWLAnnotation method*), 85
- `__hash__()` (*owlapy.owl\_axiom.OWLAnnotationAssertionAxiom method*), 86
- `__hash__()` (*owlapy.owl\_axiom.OWLAnnotationPropertyDomainAxiom method*), 87
- `__hash__()` (*owlapy.owl\_axiom.OWLAnnotationPropertyRangeAxiom method*), 87
- `__hash__()` (*owlapy.owl\_axiom.OWLClassAssertionAxiom method*), 85
- `__hash__()` (*owlapy.owl\_axiom.OWLDataPropertyCharacteristicAxiom method*), 92
- `__hash__()` (*owlapy.owl\_axiom.OWLDatatypeDefinitionAxiom method*), 79
- `__hash__()` (*owlapy.owl\_axiom.OWLDeclarationAxiom method*), 78
- `__hash__()` (*owlapy.owl\_axiom.OWLDisjointUnionAxiom method*), 84
- `__hash__()` (*owlapy.owl\_axiom.OWLHasKeyAxiom method*), 79
- `__hash__()` (*owlapy.owl\_axiom.OWLNaryClassAxiom method*), 80
- `__hash__()` (*owlapy.owl\_axiom.OWLNaryIndividualAxiom method*), 81
- `__hash__()` (*owlapy.owl\_axiom.OWLNaryPropertyAxiom method*), 82
- `__hash__()` (*owlapy.owl\_axiom.OWLObjectPropertyCharacteristicAxiom method*), 90
- `__hash__()` (*owlapy.owl\_axiom.OWLPropertyAssertionAxiom method*), 88
- `__hash__()` (*owlapy.owl\_axiom.OWLPropertyDomainAxiom method*), 92
- `__hash__()` (*owlapy.owl\_axiom.OWLPropertyRangeAxiom method*), 92
- `__hash__()` (*owlapy.owl\_axiom.OWLSubAnnotationPropertyOfAxiom method*), 86
- `__hash__()` (*owlapy.owl\_axiom.OWLSubClassOfAxiom method*), 84
- `__hash__()` (*owlapy.owl\_axiom.OWLSubPropertyAxiom method*), 87
- `__hash__()` (*owlapy.owl\_data\_ranges.OWLDataComplementOf method*), 95
- `__hash__()` (*owlapy.owl\_data\_ranges.OWLNaryDataRange method*), 94
- `__hash__()` (*owlapy.owl\_object.OWLNamedObject method*), 108
- `__hash__()` (*owlapy.owl\_object.OWLObject method*), 107
- `__hash__()` (*owlapy.owl\_ontology.Ontology method*), 112
- `__hash__()` (*owlapy.owl\_ontology.SyncOntology method*), 113
- `__hash__()` (*owlapy.owl\_property.OWLObjectInverseOf method*), 121
- `__iter__()` (*owlapy.owl\_axiom.OWLEquivalentClassesAxiom method*), 80
- `__iter__()` (*owlapy.utils.EvaluatedDescriptionSet method*), 150
- `__len__()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 98
- `__len__()` (*owlapy.owl\_ontology.Ontology method*), 110
- `__len__()` (*owlapy.owl\_ontology.SyncOntology method*), 113
- `__lt__()` (*owlapy.owl\_object.OWLNamedObject method*), 108



- \_\_lt\_\_ () (owlapy.utils.OrderedOWLObject method), 151
- \_\_repr\_\_ () (owlapy.class\_expression.class\_expression.OWLObjectComplementOf method), 38
- \_\_repr\_\_ () (owlapy.class\_expression.nary\_boolean\_expression.OWLNaryBooleanClassExpression method), 38
- \_\_repr\_\_ () (owlapy.class\_expression.OWLDataAllValuesFrom method), 65
- \_\_repr\_\_ () (owlapy.class\_expression.OWLDataCardinalityRestriction method), 61
- \_\_repr\_\_ () (owlapy.class\_expression.OWLDataHasValue method), 66
- \_\_repr\_\_ () (owlapy.class\_expression.OWLDataOneOf method), 60
- \_\_repr\_\_ () (owlapy.class\_expression.OWLDataSomeValuesFrom method), 65
- \_\_repr\_\_ () (owlapy.class\_expression.OWLDatatypeRestriction method), 63
- \_\_repr\_\_ () (owlapy.class\_expression.OWLFacetRestriction method), 64
- \_\_repr\_\_ () (owlapy.class\_expression.OWLNaryBooleanClassExpression method), 57
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectAllValuesFrom method), 62
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectCardinalityRestriction method), 59
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectComplementOf method), 55
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectHasSelf method), 60
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectHasValue method), 62
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectOneOf method), 67
- \_\_repr\_\_ () (owlapy.class\_expression.OWLObjectSomeValuesFrom method), 61
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLDataAllValuesFrom method), 50
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLDataCardinalityRestriction method), 49
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLDataHasValue method), 51
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLDataOneOf method), 51
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLDataSomeValuesFrom method), 50
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLDatatypeRestriction method), 52
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLFacetRestriction method), 52
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLObjectAllValuesFrom method), 46
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLObjectCardinalityRestriction method), 45
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLObjectHasSelf method), 47
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLObjectHasValue method), 47
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLObjectOneOf method), 48
- \_\_repr\_\_ () (owlapy.class\_expression.restriction.OWLObjectSomeValuesFrom method), 46
- \_\_repr\_\_ () (owlapy.iri.IRI method), 71
- \_\_repr\_\_ () (owlapy.namespaces.Namespaces method), 74
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLAnnotation method), 85
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLAnnotationAssertionAxiom method), 86
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLAnnotationPropertyDomainAxiom method), 87
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLAnnotationPropertyRangeAxiom method), 87
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLClassAssertionAxiom method), 85
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLDataPropertyCharacteristicAxiom method), 92
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLDatatypeDefinitionAxiom method), 79
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLDeclarationAxiom method), 78
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLDisjointUnionAxiom method), 84
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLHasKeyAxiom method), 79
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLInverseObjectPropertiesAxiom method), 83
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLNaryClassAxiom method), 80
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLNaryIndividualAxiom method), 81
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLNaryPropertyAxiom method), 82
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLObjectPropertyCharacteristicAxiom method), 90
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLPropertyAssertionAxiom method), 88
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLPropertyDomainAxiom method), 92
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLPropertyRangeAxiom method), 92
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLSubAnnotationPropertyOfAxiom method), 86
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLSubClassOfAxiom method), 84
- \_\_repr\_\_ () (owlapy.owl\_axiom.OWLSubPropertyAxiom method), 87
- \_\_repr\_\_ () (owlapy.owl\_data\_ranges.OWLDataComplementOf method), 95
- \_\_repr\_\_ () (owlapy.owl\_data\_ranges.OWLNaryDataRange method), 94
- \_\_repr\_\_ () (owlapy.owl\_object.OWLNamedObject method), 108
- \_\_repr\_\_ () (owlapy.owl\_object.OWLObject method), 107
- \_\_repr\_\_ () (owlapy.owl\_ontology.Ontology method), 112
- \_\_repr\_\_ () (owlapy.owl\_ontology.OWLOntologyID method), 110
- \_\_repr\_\_ () (owlapy.owl\_ontology.SyncOntology method), 113
- \_\_repr\_\_ () (owlapy.owl\_property.OWLObjectInverseOf method), 121
- \_\_setitem\_\_ () (owlapy.utils.LRUCache method), 152
- \_\_slots\_\_ (owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyChange attribute), 18
- \_\_slots\_\_ (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology attribute), 15
- \_\_slots\_\_ (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner attribute), 19
- \_\_slots\_\_ (owlapy.abstracts.AbstractOWLOntology attribute), 27
- \_\_slots\_\_ (owlapy.abstracts.AbstractOWLOntologyChange attribute), 27



- \_\_slots\_\_ (owlapy.abstracts.AbstractOWLReasoner attribute), 29
- \_\_slots\_\_ (owlapy.class\_expression.class\_expression.OWLBooleanClassExpression attribute), 37
- \_\_slots\_\_ (owlapy.class\_expression.class\_expression.OWLClassExpression attribute), 36
- \_\_slots\_\_ (owlapy.class\_expression.class\_expression.OWLObjectComplementOf attribute), 37
- \_\_slots\_\_ (owlapy.class\_expression.nary\_boolean\_expression.OWLNaryBooleanClassExpression attribute), 38
- \_\_slots\_\_ (owlapy.class\_expression.nary\_boolean\_expression.OWLObjectIntersectionOf attribute), 39
- \_\_slots\_\_ (owlapy.class\_expression.nary\_boolean\_expression.OWLObjectUnionOf attribute), 39
- \_\_slots\_\_ (owlapy.class\_expression.owl\_class.OWLClass attribute), 39
- \_\_slots\_\_ (owlapy.class\_expression.OWLBooleanClassExpression attribute), 55
- \_\_slots\_\_ (owlapy.class\_expression.OWLCardinalityRestriction attribute), 59
- \_\_slots\_\_ (owlapy.class\_expression.OWLClass attribute), 56
- \_\_slots\_\_ (owlapy.class\_expression.OWLClassExpression attribute), 54
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataAllValuesFrom attribute), 65
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataCardinalityRestriction attribute), 61
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataExactCardinality attribute), 66
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataHasValue attribute), 66
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataMaxCardinality attribute), 66
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataMinCardinality attribute), 66
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataRestriction attribute), 59
- \_\_slots\_\_ (owlapy.class\_expression.OWLDataSomeValuesFrom attribute), 65
- \_\_slots\_\_ (owlapy.class\_expression.OWLDatatypeRestriction attribute), 62
- \_\_slots\_\_ (owlapy.class\_expression.OWLFacetRestriction attribute), 63
- \_\_slots\_\_ (owlapy.class\_expression.OWLHasValueRestriction attribute), 58
- \_\_slots\_\_ (owlapy.class\_expression.OWLNaryBooleanClassExpression attribute), 56
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectAllValuesFrom attribute), 62
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectCardinalityRestriction attribute), 59
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectComplementOf attribute), 55
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectExactCardinality attribute), 64
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectHasSelf attribute), 60
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectHasValue attribute), 62
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectIntersectionOf attribute), 57
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectMaxCardinality attribute), 64
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectMinCardinality attribute), 64
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectOneOf attribute), 67
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectRestriction attribute), 58
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectSomeValuesFrom attribute), 61
- \_\_slots\_\_ (owlapy.class\_expression.OWLObjectUnionOf attribute), 57
- \_\_slots\_\_ (owlapy.class\_expression.OWLQuantifiedDataRestriction attribute), 60
- \_\_slots\_\_ (owlapy.class\_expression.OWLQuantifiedObjectRestriction attribute), 58
- \_\_slots\_\_ (owlapy.class\_expression.OWLQuantifiedRestriction attribute), 58
- \_\_slots\_\_ (owlapy.class\_expression.OWLRestriction attribute), 57
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLCardinalityRestriction attribute), 44
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataAllValuesFrom attribute), 50
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataCardinalityRestriction attribute), 49
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataExactCardinality attribute), 49
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataHasValue attribute), 51
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataMaxCardinality attribute), 49
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataMinCardinality attribute), 49
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataRestriction attribute), 48
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDataSomeValuesFrom attribute), 50
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLDatatypeRestriction attribute), 52
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLFacetRestriction attribute), 52
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLHasValueRestriction attribute), 43
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectAllValuesFrom attribute), 46
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectCardinalityRestriction attribute), 45
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectExactCardinality attribute), 45
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectHasSelf attribute), 47
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectHasValue attribute), 47
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectMaxCardinality attribute), 45
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectMinCardinality attribute), 45
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectOneOf attribute), 47
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectRestriction attribute), 43
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLObjectSomeValuesFrom attribute), 46
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLQuantifiedDataRestriction attribute), 48
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLQuantifiedObjectRestriction attribute), 44
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLQuantifiedRestriction attribute), 44
- \_\_slots\_\_ (owlapy.class\_expression.restriction.OWLRestriction attribute), 43
- \_\_slots\_\_ (owlapy.converter.Owl2SparqlConverter attribute), 68

- \_\_slots\_\_ (owlapy.converter.VariablesMapping attribute), 68
- \_\_slots\_\_ (owlapy.iri.IRI attribute), 71
- \_\_slots\_\_ (owlapy.meta\_classes.HasCardinality attribute), 73
- \_\_slots\_\_ (owlapy.meta\_classes.HasFiller attribute), 73
- \_\_slots\_\_ (owlapy.meta\_classes.HasIRI attribute), 72
- \_\_slots\_\_ (owlapy.meta\_classes.HasOperands attribute), 72
- \_\_slots\_\_ (owlapy.namespaces.Namespaces attribute), 73
- \_\_slots\_\_ (owlapy.OntologyManager attribute), 156
- \_\_slots\_\_ (owlapy.owl\_annotation.OWLAnnotationObject attribute), 74
- \_\_slots\_\_ (owlapy.owl\_annotation.OWLAnnotationSubject attribute), 74
- \_\_slots\_\_ (owlapy.owl\_annotation.OWLAnnotationValue attribute), 75
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAnnotation attribute), 85
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAnnotationAssertionAxiom attribute), 86
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAnnotationAxiom attribute), 85
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAnnotationProperty attribute), 85
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAnnotationPropertyDomainAxiom attribute), 87
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAnnotationPropertyRangeAxiom attribute), 87
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLAsymmetricObjectPropertyAxiom attribute), 90
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLClassAxiom attribute), 77
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLClassAssertionAxiom attribute), 84
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLClassAxiom attribute), 78
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDataPropertyAssertionAxiom attribute), 89
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDataPropertyAxiom attribute), 78
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDataPropertyCharacteristicAxiom attribute), 91
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDataPropertyDomainAxiom attribute), 93
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDataPropertyRangeAxiom attribute), 94
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDatatypeDefinitionAxiom attribute), 79
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDeclarationAxiom attribute), 78
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDifferentIndividualsAxiom attribute), 81
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDisjointClassesAxiom attribute), 81
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDisjointDataPropertiesAxiom attribute), 83
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDisjointObjectPropertiesAxiom attribute), 82
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLDisjointUnionAxiom attribute), 84
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLEquivalentClassesAxiom attribute), 80
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLEquivalentDataPropertiesAxiom attribute), 83
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLEquivalentObjectPropertiesAxiom attribute), 82
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLFunctionalDataPropertyAxiom attribute), 92
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLFunctionalObjectPropertyAxiom attribute), 90
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLHasKeyAxiom attribute), 79
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLIndividualAxiom attribute), 78
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLInverseFunctionalObjectPropertyAxiom attribute), 90
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLInverseObjectPropertiesAxiom attribute), 83
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLIrreflexiveObjectPropertyAxiom attribute), 91
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLLogicalAxiom attribute), 77
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLNaryAxiom attribute), 80
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLNaryClassAxiom attribute), 80
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLNaryIndividualAxiom attribute), 81
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLNaryPropertyAxiom attribute), 82
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLNegativeDataPropertyAssertionAxiom attribute), 89
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLNegativeObjectPropertyAssertionAxiom attribute), 89
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLObjectPropertyAssertionAxiom attribute), 89
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLObjectPropertyAxiom attribute), 78
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLObjectPropertyCharacteristicAxiom attribute), 90
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLObjectPropertyDomainAxiom attribute), 93
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLObjectPropertyRangeAxiom attribute), 93
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLPropertyAssertionAxiom attribute), 88
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLPropertyAxiom attribute), 78
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLPropertyDomainAxiom attribute), 92
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLPropertyRangeAxiom attribute), 92
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLReflexiveObjectPropertyAxiom attribute), 91
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSameIndividualAxiom attribute), 82
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSubAnnotationPropertyOfAxiom attribute), 86
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSubClassOfAxiom attribute), 84
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSubDataPropertyOfAxiom attribute), 88
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSubObjectPropertyOfAxiom attribute), 88
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSubPropertyAxiom attribute), 87
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLSymmetricObjectPropertyAxiom attribute), 91
- \_\_slots\_\_ (owlapy.owl\_axiom.OWLTransitiveObjectPropertyAxiom attribute), 91

- `__slots__` (*owlapy.owl\_axiom.OWLUnaryPropertyAxiom attribute*), 89
- `__slots__` (*owlapy.owl\_data\_ranges.OWLDataIntersectionOf attribute*), 95
- `__slots__` (*owlapy.owl\_data\_ranges.OWLDataUnionOf attribute*), 95
- `__slots__` (*owlapy.owl\_data\_ranges.OWLNaryDataRange attribute*), 94
- `__slots__` (*owlapy.owl\_datatype.OWLDatatype attribute*), 96
- `__slots__` (*owlapy.owl\_hierarchy.AbstractHierarchy attribute*), 96
- `__slots__` (*owlapy.owl\_individual.OWLIndividual attribute*), 100
- `__slots__` (*owlapy.owl\_individual.OWLNamedIndividual attribute*), 100
- `__slots__` (*owlapy.owl\_literal.OWLLiteral attribute*), 104
- `__slots__` (*owlapy.owl\_object.OWLEntity attribute*), 108
- `__slots__` (*owlapy.owl\_object.OWLNamedObject attribute*), 108
- `__slots__` (*owlapy.owl\_object.OWLObject attribute*), 107
- `__slots__` (*owlapy.owl\_ontology\_manager.AddImport attribute*), 116
- `__slots__` (*owlapy.owl\_ontology\_manager.OntologyManager attribute*), 116
- `__slots__` (*owlapy.owl\_ontology\_manager.OWLImportsDeclaration attribute*), 116
- `__slots__` (*owlapy.owl\_ontology.FromOwready2 attribute*), 115
- `__slots__` (*owlapy.owl\_ontology.Ontology attribute*), 110
- `__slots__` (*owlapy.owl\_ontology.OWLOntologyID attribute*), 109
- `__slots__` (*owlapy.owl\_ontology.ToOwready2 attribute*), 115
- `__slots__` (*owlapy.owl\_property.OWLDataProperty attribute*), 121
- `__slots__` (*owlapy.owl\_property.OWLDataPropertyExpression attribute*), 119
- `__slots__` (*owlapy.owl\_property.OWLObjectInverseOf attribute*), 121
- `__slots__` (*owlapy.owl\_property.OWLObjectProperty attribute*), 120
- `__slots__` (*owlapy.owl\_property.OWLObjectPropertyExpression attribute*), 119
- `__slots__` (*owlapy.owl\_property.OWLProperty attribute*), 120
- `__slots__` (*owlapy.owl\_property.OWLPropertyExpression attribute*), 118
- `__slots__` (*owlapy.render.DLSyntaxObjectRenderer attribute*), 143
- `__slots__` (*owlapy.render.ManchesterOWLSyntaxOWLObjectRenderer attribute*), 143
- `__slots__` (*owlapy.utils.EvaluatedDescriptionSet attribute*), 150
- `__slots__` (*owlapy.utils.OrderedOWLObject attribute*), 150
- `__slots__` (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 148
- `__str__` () (*owlapy.owl\_literal.FloatSpecialValue method*), 104

## A

- `AbstractHierarchy` (*class in owlapy.owl\_hierarchy*), 96
- `AbstractOWLOntology` (*class in owlapy.abstracts*), 27
- `AbstractOWLOntology` (*class in owlapy.abstracts.abstract\_owl\_ontology*), 15
- `AbstractOWLOntologyChange` (*class in owlapy.abstracts*), 27
- `AbstractOWLOntologyChange` (*class in owlapy.abstracts.abstract\_owl\_ontology\_manager*), 18
- `AbstractOWLOntologyManager` (*class in owlapy.abstracts*), 26
- `AbstractOWLOntologyManager` (*class in owlapy.abstracts.abstract\_owl\_ontology\_manager*), 18
- `AbstractOWLReasoner` (*class in owlapy.abstracts*), 29
- `AbstractOWLReasoner` (*class in owlapy.abstracts.abstract\_owl\_reasoner*), 19
- `add_axiom` () (*owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method*), 17
- `add_axiom` () (*owlapy.abstracts.AbstractOWLOntology method*), 29
- `add_axiom` () (*owlapy.owl\_ontology.Ontology method*), 112
- `add_axiom` () (*owlapy.owl\_ontology.SyncOntology method*), 115
- `AddImport` (*class in owlapy.owl\_ontology\_manager*), 116
- `all_data_property_values` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method*), 25
- `all_data_property_values` () (*owlapy.abstracts.AbstractOWLReasoner method*), 35
- `all_data_property_values` () (*owlapy.owl\_reasoner.StructuralReasoner method*), 124
- `annotations` () (*owlapy.owl\_axiom.OWLAxiom method*), 77
- `append` () (*owlapy.converter.Owl2SparqlConverter method*), 69
- `append_triple` () (*owlapy.converter.Owl2SparqlConverter method*), 69
- `apply_change` () (*owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyManager method*), 18
- `apply_change` () (*owlapy.abstracts.AbstractOWLOntologyManager method*), 26
- `apply_change` () (*owlapy.OntologyManager method*), 156
- `apply_change` () (*owlapy.owl\_ontology\_manager.OntologyManager method*), 117
- `apply_change` () (*owlapy.owl\_ontology\_manager.SyncOntologyManager method*), 118
- `as_anonymous_individual` () (*owlapy.owl\_annotation.OWLAnnotationObject method*), 74
- `as_confusion_matrix_query` () (*owlapy.converter.Owl2SparqlConverter method*), 70
- `as_index` () (*in module owlapy.utils*), 151
- `as_intersection_of_min_max` () (*owlapy.class\_expression.OWLDataExactCardinality method*), 67
- `as_intersection_of_min_max` () (*owlapy.class\_expression.OWLObjectExactCardinality method*), 64
- `as_intersection_of_min_max` () (*owlapy.class\_expression.restriction.OWLDataExactCardinality method*), 50
- `as_intersection_of_min_max` () (*owlapy.class\_expression.restriction.OWLObjectExactCardinality method*), 46
- `as_iri` () (*owlapy.iri.IRI method*), 71

as\_iri() (*owlapy.owl\_annotation.OWLAnnotationObject method*), 74  
 as\_literal() (*owlapy.owl\_annotation.OWLAnnotationValue method*), 75  
 as\_literal() (*owlapy.owl\_literal.OWLLiteral method*), 107  
 as\_object\_union\_of() (*owlapy.class\_expression.OWLObjectOneOf method*), 67  
 as\_object\_union\_of() (*owlapy.class\_expression.restriction.OWLObjectOneOf method*), 48  
 as\_pairwise\_axioms() (*owlapy.owl\_axiom.OWLNaryAxiom method*), 80  
 as\_pairwise\_axioms() (*owlapy.owl\_axiom.OWLNaryClassAxiom method*), 80  
 as\_pairwise\_axioms() (*owlapy.owl\_axiom.OWLNaryIndividualAxiom method*), 81  
 as\_pairwise\_axioms() (*owlapy.owl\_axiom.OWLNaryPropertyAxiom method*), 82  
 as\_query() (*owlapy.converter.Owl2SparqlConverter method*), 70  
 as\_some\_values\_from() (*owlapy.class\_expression.OWLDataHasValue method*), 66  
 as\_some\_values\_from() (*owlapy.class\_expression.OWLObjectHasValue method*), 62  
 as\_some\_values\_from() (*owlapy.class\_expression.restriction.OWLDataHasValue method*), 51  
 as\_some\_values\_from() (*owlapy.class\_expression.restriction.OWLObjectHasValue method*), 47  
 as\_str() (*owlapy.iri.IRI method*), 71

## B

best() (*owlapy.utils.EvaluatedDescriptionSet method*), 150  
 best\_quality\_value() (*owlapy.utils.EvaluatedDescriptionSet method*), 150  
 BOOLEAN (*owlapy.vocab.XSDVocabulary attribute*), 153  
 BooleanOWLDatatype (*in module owlapy.owl\_literal*), 103

## C

cache (*owlapy.utils.LRUCache attribute*), 152  
 cache\_clear() (*owlapy.utils.LRUCache method*), 152  
 cache\_get (*owlapy.utils.LRUCache attribute*), 152  
 cache\_info() (*owlapy.utils.LRUCache method*), 152  
 cache\_len (*owlapy.utils.LRUCache attribute*), 152  
 ce (*owlapy.converter.Owl2SparqlConverter attribute*), 68  
 children() (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 97  
 class\_cache (*owlapy.owl\_reasoner.StructuralReasoner attribute*), 122  
 class\_cnt (*owlapy.converter.VariablesMapping attribute*), 68  
 class\_expressions() (*owlapy.owl\_axiom.OWLNaryClassAxiom method*), 80  
 class\_length (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 148  
 classes\_in\_signature() (*owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method*), 16  
 classes\_in\_signature() (*owlapy.abstracts.AbstractOWLOntology method*), 27  
 classes\_in\_signature() (*owlapy.owl\_ontology.Ontology method*), 110  
 classes\_in\_signature() (*owlapy.owl\_ontology.SyncOntology method*), 113  
 ClassHierarchy (*class in owlapy.owl\_hierarchy*), 98  
 clean() (*owlapy.utils.EvaluatedDescriptionSet method*), 150  
 cnt (*owlapy.converter.Owl2SparqlConverter attribute*), 69  
 combine\_nary\_expressions() (*in module owlapy.utils*), 151  
 concept\_reducer() (*in module owlapy.utils*), 147  
 concept\_reducer\_properties() (*in module owlapy.utils*), 147  
 ConceptOperandSorter (*class in owlapy.utils*), 150  
 contains\_named\_equivalent\_class() (*owlapy.owl\_axiom.OWLEquivalentClassesAxiom method*), 80  
 contains\_owl\_nothing() (*owlapy.owl\_axiom.OWLEquivalentClassesAxiom method*), 80  
 contains\_owl\_thing() (*owlapy.owl\_axiom.OWLEquivalentClassesAxiom method*), 81  
 convert() (*owlapy.converter.Owl2SparqlConverter method*), 69  
 converter (*in module owlapy.converter*), 70  
 create() (*owlapy.iri.IRI static method*), 71  
 create\_ontology() (*in module owlapy.static\_funcs*), 144  
 create\_ontology() (*owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyManager method*), 18  
 create\_ontology() (*owlapy.abstracts.AbstractOWLOntologyManager method*), 26  
 create\_ontology() (*owlapy.OntologyManager method*), 156  
 create\_ontology() (*owlapy.owl\_ontology\_manager.OntologyManager method*), 116  
 create\_ontology() (*owlapy.owl\_ontology\_manager.SyncOntologyManager method*), 117  
 csv\_to\_rdf\_kg() (*in module owlapy.util\_owl\_static\_funcs*), 145  
 current\_variable (*owlapy.converter.Owl2SparqlConverter property*), 69

## D

data\_all\_values\_length (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149  
 data\_cardinality\_length (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149  
 data\_complement\_length (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149  
 data\_has\_value\_length (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149  
 data\_intersection\_length (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149



`data_one_of_length` (*owlapy.utils.OWLClassExpressionLengthMetric* attribute), 149  
`data_properties_in_signature` () (*owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology* method), 16  
`data_properties_in_signature` () (*owlapy.abstracts.AbstractOWLOntology* method), 27  
`data_properties_in_signature` () (*owlapy.owl\_ontology.Ontology* method), 110  
`data_properties_in_signature` () (*owlapy.owl\_ontology.SyncOntology* method), 113  
`data_property_domain_axioms` () (*owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology* method), 16  
`data_property_domain_axioms` () (*owlapy.abstracts.AbstractOWLOntology* method), 28  
`data_property_domain_axioms` () (*owlapy.owl\_ontology.Ontology* method), 111  
`data_property_domain_axioms` () (*owlapy.owl\_ontology.SyncOntology* method), 113  
`data_property_domains` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 19  
`data_property_domains` () (*owlapy.abstracts.AbstractOWLReasoner* method), 29  
`data_property_domains` () (*owlapy.owl\_reasoner.StructuralReasoner* method), 122  
`data_property_domains` () (*owlapy.owl\_reasoner.SyncReasoner* method), 129  
`data_property_length` (*owlapy.utils.OWLClassExpressionLengthMetric* attribute), 149  
`data_property_range_axioms` () (*owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology* method), 16  
`data_property_range_axioms` () (*owlapy.abstracts.AbstractOWLOntology* method), 28  
`data_property_range_axioms` () (*owlapy.owl\_ontology.Ontology* method), 111  
`data_property_range_axioms` () (*owlapy.owl\_ontology.SyncOntology* method), 114  
`data_property_ranges` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 25  
`data_property_ranges` () (*owlapy.abstracts.AbstractOWLReasoner* method), 35  
`data_property_values` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 21  
`data_property_values` () (*owlapy.abstracts.AbstractOWLReasoner* method), 31  
`data_property_values` () (*owlapy.owl\_reasoner.StructuralReasoner* method), 124  
`data_property_values` () (*owlapy.owl\_reasoner.SyncReasoner* method), 132  
`data_some_values_length` (*owlapy.utils.OWLClassExpressionLengthMetric* attribute), 149  
`data_union_length` (*owlapy.utils.OWLClassExpressionLengthMetric* attribute), 149  
`datatype_length` (*owlapy.utils.OWLClassExpressionLengthMetric* attribute), 149  
`DatatypePropertyHierarchy` (class in *owlapy.owl\_hierarchy*), 99  
`DATE` (*owlapy.vocab.XSDVocabulary* attribute), 153  
`DATE_TIME` (*owlapy.vocab.XSDVocabulary* attribute), 154  
`DATE_TIME_STAMP` (*owlapy.vocab.XSDVocabulary* attribute), 154  
`DateOWLDatatype` (in module *owlapy.owl\_literal*), 103  
`DateTimeOWLDatatype` (in module *owlapy.owl\_literal*), 103  
`DECIMAL` (*owlapy.vocab.XSDVocabulary* attribute), 153  
`DecimalOWLDatatype` (in module *owlapy.owl\_literal*), 103  
`dict` (*owlapy.converter.VariablesMapping* attribute), 68  
`different_individuals` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 21  
`different_individuals` () (*owlapy.abstracts.AbstractOWLReasoner* method), 31  
`different_individuals` () (*owlapy.owl\_reasoner.StructuralReasoner* method), 124  
`different_individuals` () (*owlapy.owl\_reasoner.SyncReasoner* method), 131  
`disjoint_classes` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 20  
`disjoint_classes` () (*owlapy.abstracts.AbstractOWLReasoner* method), 30  
`disjoint_classes` () (*owlapy.owl\_reasoner.StructuralReasoner* method), 124  
`disjoint_classes` () (*owlapy.owl\_reasoner.SyncReasoner* method), 129  
`disjoint_data_properties` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 23  
`disjoint_data_properties` () (*owlapy.abstracts.AbstractOWLReasoner* method), 33  
`disjoint_data_properties` () (*owlapy.owl\_reasoner.StructuralReasoner* method), 126  
`disjoint_data_properties` () (*owlapy.owl\_reasoner.SyncReasoner* method), 133  
`disjoint_object_properties` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 23  
`disjoint_object_properties` () (*owlapy.abstracts.AbstractOWLReasoner* method), 32  
`disjoint_object_properties` () (*owlapy.owl\_reasoner.StructuralReasoner* method), 126  
`disjoint_object_properties` () (*owlapy.owl\_reasoner.SyncReasoner* method), 132  
`DL_GRAMMAR` (in module *owlapy.parser*), 138  
`dl_to_owl_expression` () (in module *owlapy*), 155  
`dl_to_owl_expression` () (in module *owlapy.parser*), 141  
`DLparser` (in module *owlapy.parser*), 140  
`DLrenderer` (in module *owlapy.render*), 143  
`DLSyntaxObjectRenderer` (class in *owlapy.render*), 143  
`DLSyntaxParser` (class in *owlapy.parser*), 138  
`DOUBLE` (*owlapy.vocab.XSDVocabulary* attribute), 153  
`DoubleOWLDatatype` (in module *owlapy.owl\_literal*), 103  
`download_external_files` () (in module *owlapy.static\_funcs*), 144  
`DURATION` (*owlapy.vocab.XSDVocabulary* attribute), 154  
`DurationOWLDatatype` (in module *owlapy.owl\_literal*), 103

## E

`equivalent_classes` () (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner* method), 20

equivalent\_classes() (owlapy.abstracts.AbstractOWLReasoner method), 30  
 equivalent\_classes() (owlapy.owl\_reasoner.StructuralReasoner method), 123  
 equivalent\_classes() (owlapy.owl\_reasoner.SyncReasoner method), 128  
 equivalent\_classes\_axioms() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 16  
 equivalent\_classes\_axioms() (owlapy.abstracts.AbstractOWLOntology method), 27  
 equivalent\_classes\_axioms() (owlapy.owl\_ontology.Ontology method), 110  
 equivalent\_classes\_axioms() (owlapy.owl\_ontology.SyncOntology method), 113  
 equivalent\_data\_properties() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 21  
 equivalent\_data\_properties() (owlapy.abstracts.AbstractOWLReasoner method), 31  
 equivalent\_data\_properties() (owlapy.owl\_reasoner.StructuralReasoner method), 126  
 equivalent\_data\_properties() (owlapy.owl\_reasoner.SyncReasoner method), 132  
 equivalent\_object\_properties() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 21  
 equivalent\_object\_properties() (owlapy.abstracts.AbstractOWLReasoner method), 31  
 equivalent\_object\_properties() (owlapy.owl\_reasoner.StructuralReasoner method), 126  
 equivalent\_object\_properties() (owlapy.owl\_reasoner.SyncReasoner method), 132  
 EvaluatedDescriptionSet (class in owlapy.utils), 149

## F

FLOAT (owlapy.vocab.XSDVocabulary attribute), 153  
 FloatOWLDatatype (in module owlapy.owl\_literal), 103  
 FloatSpecialValue (class in owlapy.owl\_literal), 104  
 for\_all\_de\_morgan (owlapy.converter.Owl2SparqlConverter attribute), 69  
 forAll() (owlapy.converter.Owl2SparqlConverter method), 69  
 forAllDeMorgan() (owlapy.converter.Owl2SparqlConverter method), 69  
 FRACTION\_DIGITS (owlapy.class\_expression.OWLFacet attribute), 63  
 FRACTION\_DIGITS (owlapy.vocab.OWLFacet attribute), 154  
 from\_str() (owlapy.class\_expression.OWLFacet static method), 63  
 from\_str() (owlapy.vocab.OWLFacet static method), 154  
 FromOwlready2 (class in owlapy.owl\_ontology), 115  
 full (owlapy.utils.LRUCache attribute), 152

## G

GDAY (owlapy.vocab.XSDVocabulary attribute), 154  
 GDayOWLDatatype (in module owlapy.owl\_literal), 103  
 general\_class\_axioms() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 16  
 general\_class\_axioms() (owlapy.abstracts.AbstractOWLOntology method), 28  
 general\_class\_axioms() (owlapy.owl\_ontology.Ontology method), 111  
 general\_class\_axioms() (owlapy.owl\_ontology.SyncOntology method), 113  
 generate\_and\_save\_inferred\_class\_assertion\_axioms() (owlapy.owl\_reasoner.SyncReasoner method), 134  
 generic\_visit() (owlapy.parser.DLSyntaxParser method), 140  
 generic\_visit() (owlapy.parser.ManchesterOWLSyntaxParser method), 138  
 get\_abox\_axioms() (owlapy.owl\_ontology.Ontology method), 110  
 get\_abox\_axioms() (owlapy.owl\_ontology.SyncOntology method), 114  
 get\_abox\_axioms\_between\_individuals() (owlapy.owl\_ontology.Ontology method), 110  
 get\_abox\_axioms\_between\_individuals\_and\_classes() (owlapy.owl\_ontology.Ontology method), 110  
 get\_bottom\_entity() (owlapy.owl\_hierarchy.AbstractHierarchy class method), 97  
 get\_bottom\_entity() (owlapy.owl\_hierarchy.ClassHierarchy class method), 98  
 get\_bottom\_entity() (owlapy.owl\_hierarchy.DatatypePropertyHierarchy class method), 99  
 get\_bottom\_entity() (owlapy.owl\_hierarchy.ObjectPropertyHierarchy class method), 99  
 get\_cardinality() (owlapy.class\_expression.OWLCardinalityRestriction method), 59  
 get\_cardinality() (owlapy.class\_expression.restriction.OWLCardinalityRestriction method), 44  
 get\_cardinality() (owlapy.meta\_classes.HasCardinality method), 73  
 get\_class\_expression() (owlapy.owl\_axiom.OWLClassAssertionAxiom method), 84  
 get\_class\_expression() (owlapy.owl\_axiom.OWLHasKeyAxiom method), 79  
 get\_class\_expressions() (owlapy.owl\_axiom.OWLDisjointUnionAxiom method), 84  
 get\_class\_nnf() (owlapy.utils.NNF method), 151  
 get\_data\_range() (owlapy.owl\_data\_ranges.OWLDataComplementOf method), 95  
 get\_data\_range() (owlapy.owl\_axiom.OWLDatatypeDefinitionAxiom method), 79  
 get\_datatype() (owlapy.class\_expression.OWLDatatypeRestriction method), 63  
 get\_datatype() (owlapy.class\_expression.restriction.OWLDatatypeRestriction method), 52  
 get\_datatype() (owlapy.owl\_axiom.OWLDatatypeDefinitionAxiom method), 79  
 get\_datatype() (owlapy.owl\_literal.OWLLiteral method), 107  
 get\_default() (owlapy.utils.OWLClassExpressionLengthMetric static method), 149  
 get\_default\_document\_iri() (owlapy.owl\_ontology.OWLOntologyID method), 109  
 get\_domain() (owlapy.owl\_axiom.OWLAnnotationPropertyDomainAxiom method), 87  
 get\_domain() (owlapy.owl\_axiom.OWLPropertyDomainAxiom method), 92  
 get\_entity() (owlapy.owl\_axiom.OWLDeclarationAxiom method), 78

`get_expression_length()` (in module `owlapy.utils`), 149  
`get_facet()` (`owlapy.class_expression.OwlFacetRestriction` method), 63  
`get_facet()` (`owlapy.class_expression.restriction.OwlFacetRestriction` method), 52  
`get_facet_restrictions()` (`owlapy.class_expression.OwlDatatypeRestriction` method), 63  
`get_facet_restrictions()` (`owlapy.class_expression.restriction.OwlDatatypeRestriction` method), 52  
`get_facet_value()` (`owlapy.class_expression.OwlFacetRestriction` method), 63  
`get_facet_value()` (`owlapy.class_expression.restriction.OwlFacetRestriction` method), 52  
`get_filler()` (`owlapy.class_expression.OwlCardinalityRestriction` method), 59  
`get_filler()` (`owlapy.class_expression.OwlHasValueRestriction` method), 58  
`get_filler()` (`owlapy.class_expression.OwlQuantifiedDataRestriction` method), 61  
`get_filler()` (`owlapy.class_expression.OwlQuantifiedObjectRestriction` method), 58  
`get_filler()` (`owlapy.class_expression.restriction.OwlCardinalityRestriction` method), 44  
`get_filler()` (`owlapy.class_expression.restriction.OwlHasValueRestriction` method), 43  
`get_filler()` (`owlapy.class_expression.restriction.OwlQuantifiedDataRestriction` method), 48  
`get_filler()` (`owlapy.class_expression.restriction.OwlQuantifiedObjectRestriction` method), 44  
`get_filler()` (`owlapy.meta_classes.HasFiller` method), 73  
`get_first_property()` (`owlapy.owl_axiom.OwlInverseObjectPropertiesAxiom` method), 83  
`get_import_declaration()` (`owlapy.owl_ontology_manager.AddImport` method), 116  
`get_individual()` (`owlapy.owl_axiom.OwlClassAssertionAxiom` method), 84  
`get_instances_from_owl_class()` (`owlapy.owl_reasoner.StructuralReasoner` method), 128  
`get_inverse()` (`owlapy.owl_property.OwlObjectInverseOf` method), 121  
`get_inverse_property()` (`owlapy.owl_property.OwlObjectInverseOf` method), 121  
`get_inverse_property()` (`owlapy.owl_property.OwlObjectProperty` method), 120  
`get_inverse_property()` (`owlapy.owl_property.OwlObjectPropertyExpression` method), 119  
`get_literal()` (`owlapy.owl_literal.OwlLiteral` method), 104  
`get_named_property()` (`owlapy.owl_property.OwlObjectInverseOf` method), 121  
`get_named_property()` (`owlapy.owl_property.OwlObjectProperty` method), 120  
`get_named_property()` (`owlapy.owl_property.OwlObjectPropertyExpression` method), 119  
`get_namespace()` (`owlapy.iri.IRI` method), 71  
`get_nnf()` (`owlapy.class_expression.class_expression.OwlAnonymousClassExpression` method), 37  
`get_nnf()` (`owlapy.class_expression.class_expression.OwlClassExpression` method), 37  
`get_nnf()` (`owlapy.class_expression.owl_class.OwlClass` method), 40  
`get_nnf()` (`owlapy.class_expression.OwlAnonymousClassExpression` method), 55  
`get_nnf()` (`owlapy.class_expression.OwlClass` method), 56  
`get_nnf()` (`owlapy.class_expression.OwlClassExpression` method), 54  
`get_object()` (`owlapy.owl_axiom.OwlPropertyAssertionAxiom` method), 88  
`get_object_complement_of()` (`owlapy.class_expression.class_expression.OwlAnonymousClassExpression` method), 37  
`get_object_complement_of()` (`owlapy.class_expression.class_expression.OwlClassExpression` method), 37  
`get_object_complement_of()` (`owlapy.class_expression.owl_class.OwlClass` method), 40  
`get_object_complement_of()` (`owlapy.class_expression.OwlAnonymousClassExpression` method), 55  
`get_object_complement_of()` (`owlapy.class_expression.OwlClass` method), 56  
`get_object_complement_of()` (`owlapy.class_expression.OwlClassExpression` method), 54  
`get_ontology()` (`owlapy.abstracts.abstract_owl_ontology_manager.AbstractOwlOntologyChange` method), 18  
`get_ontology()` (`owlapy.abstracts.AbstractOwlOntologyChange` method), 27  
`get_ontology_id()` (`owlapy.abstracts.abstract_owl_ontology_manager.AbstractOwlOntology` method), 17  
`get_ontology_id()` (`owlapy.abstracts.AbstractOwlOntology` method), 28  
`get_ontology_id()` (`owlapy.owl_ontology.Ontology` method), 111  
`get_ontology_id()` (`owlapy.owl_ontology.SyncOntology` method), 115  
`get_ontology_iri()` (`owlapy.owl_ontology.OwlOntologyID` method), 109  
`get_operand()` (`owlapy.class_expression.class_expression.OwlObjectComplementOf` method), 38  
`get_operand()` (`owlapy.class_expression.OwlObjectComplementOf` method), 55  
`get_original_iri()` (`owlapy.owl_ontology.Ontology` method), 112  
`get_owl_class()` (`owlapy.owl_axiom.OwlDisjointUnionAxiom` method), 84  
`get_owl_disjoint_classes_axiom()` (`owlapy.owl_axiom.OwlDisjointUnionAxiom` method), 84  
`get_owl_equivalent_classes_axiom()` (`owlapy.owl_axiom.OwlDisjointUnionAxiom` method), 84  
`get_owl_ontology_manager()` (`owlapy.abstracts.abstract_owl_ontology_manager.AbstractOwlOntology` method), 17  
`get_owl_ontology_manager()` (`owlapy.abstracts.AbstractOwlOntology` method), 28  
`get_owl_ontology_manager()` (`owlapy.owl_ontology.Ontology` method), 111  
`get_owl_ontology_manager()` (`owlapy.owl_ontology.SyncOntology` method), 114  
`get_owlapi_manager()` (`owlapy.owl_ontology_manager.SyncOntologyManager` method), 118  
`get_owlapi_ontology()` (`owlapy.owl_ontology.SyncOntology` method), 115  
`get_property()` (`owlapy.class_expression.OwlDataAllValuesFrom` method), 65  
`get_property()` (`owlapy.class_expression.OwlDataCardinalityRestriction` method), 61  
`get_property()` (`owlapy.class_expression.OwlDataHasValue` method), 66  
`get_property()` (`owlapy.class_expression.OwlDataSomeValuesFrom` method), 65  
`get_property()` (`owlapy.class_expression.OwlObjectAllValuesFrom` method), 62  
`get_property()` (`owlapy.class_expression.OwlObjectCardinalityRestriction` method), 59  
`get_property()` (`owlapy.class_expression.OwlObjectHasSelf` method), 60

`get_property()` (*owlapy.class\_expression.OwlObjectHasValue method*), 62  
`get_property()` (*owlapy.class\_expression.OwlObjectRestriction method*), 58  
`get_property()` (*owlapy.class\_expression.OwlObjectSomeValuesFrom method*), 61  
`get_property()` (*owlapy.class\_expression.OwlRestriction method*), 57  
`get_property()` (*owlapy.class\_expression.restriction.OwlDataAllValuesFrom method*), 50  
`get_property()` (*owlapy.class\_expression.restriction.OwlDataCardinalityRestriction method*), 49  
`get_property()` (*owlapy.class\_expression.restriction.OwlDataHasValue method*), 51  
`get_property()` (*owlapy.class\_expression.restriction.OwlDataSomeValuesFrom method*), 50  
`get_property()` (*owlapy.class\_expression.restriction.OwlObjectAllValuesFrom method*), 46  
`get_property()` (*owlapy.class\_expression.restriction.OwlObjectCardinalityRestriction method*), 45  
`get_property()` (*owlapy.class\_expression.restriction.OwlObjectHasSelf method*), 47  
`get_property()` (*owlapy.class\_expression.restriction.OwlObjectHasValue method*), 47  
`get_property()` (*owlapy.class\_expression.restriction.OwlObjectRestriction method*), 43  
`get_property()` (*owlapy.class\_expression.restriction.OwlObjectSomeValuesFrom method*), 46  
`get_property()` (*owlapy.class\_expression.restriction.OwlRestriction method*), 43  
`get_property()` (*owlapy.owl\_axiom.OwlAnnotation method*), 85  
`get_property()` (*owlapy.owl\_axiom.OwlAnnotationAssertionAxiom method*), 86  
`get_property()` (*owlapy.owl\_axiom.OwlAnnotationPropertyDomainAxiom method*), 87  
`get_property()` (*owlapy.owl\_axiom.OwlAnnotationPropertyRangeAxiom method*), 87  
`get_property()` (*owlapy.owl\_axiom.OwlPropertyAssertionAxiom method*), 88  
`get_property()` (*owlapy.owl\_axiom.OwlUnaryPropertyAxiom method*), 90  
`get_property_expressions()` (*owlapy.owl\_axiom.OwlHasKeyAxiom method*), 79  
`get_range()` (*owlapy.owl\_axiom.OwlAnnotationPropertyRangeAxiom method*), 87  
`get_range()` (*owlapy.owl\_axiom.OwlPropertyRangeAxiom method*), 92  
`get_remainder()` (*owlapy.iri.IRI method*), 72  
`get_root_ontology()` (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOwlReasoner method*), 24  
`get_root_ontology()` (*owlapy.abstracts.AbstractOwlReasoner method*), 34  
`get_root_ontology()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 128  
`get_root_ontology()` (*owlapy.owl\_reasoner.SyncReasoner method*), 135  
`get_second_property()` (*owlapy.owl\_axiom.OwlInverseObjectPropertiesAxiom method*), 83  
`get_signature()` (*owlapy.owl\_ontology.SyncOntology method*), 114  
`get_sub_class()` (*owlapy.owl\_axiom.OwlSubClassOfAxiom method*), 84  
`get_sub_property()` (*owlapy.owl\_axiom.OwlSubAnnotationPropertyOfAxiom method*), 86  
`get_sub_property()` (*owlapy.owl\_axiom.OwlSubPropertyAxiom method*), 87  
`get_subject()` (*owlapy.owl\_axiom.OwlAnnotationAssertionAxiom method*), 86  
`get_subject()` (*owlapy.owl\_axiom.OwlPropertyAssertionAxiom method*), 88  
`get_super_class()` (*owlapy.owl\_axiom.OwlSubClassOfAxiom method*), 84  
`get_super_property()` (*owlapy.owl\_axiom.OwlSubAnnotationPropertyOfAxiom method*), 86  
`get_super_property()` (*owlapy.owl\_axiom.OwlSubPropertyAxiom method*), 87  
`get_tbox_axioms()` (*owlapy.owl\_ontology.Ontology method*), 110  
`get_tbox_axioms()` (*owlapy.owl\_ontology.SyncOntology method*), 114  
`get_top_entity()` (*owlapy.owl\_hierarchy.AbstractHierarchy class method*), 97  
`get_top_entity()` (*owlapy.owl\_hierarchy.ClassHierarchy class method*), 98  
`get_top_entity()` (*owlapy.owl\_hierarchy.DatatypePropertyHierarchy class method*), 99  
`get_top_entity()` (*owlapy.owl\_hierarchy.ObjectPropertyHierarchy class method*), 99  
`get_top_level_cnf()` (*owlapy.utils.TopLevelCNF method*), 151  
`get_top_level_dnf()` (*owlapy.utils.TopLevelDNF method*), 151  
`get_value()` (*owlapy.owl\_axiom.OwlAnnotation method*), 85  
`get_value()` (*owlapy.owl\_axiom.OwlAnnotationAssertionAxiom method*), 86  
`get_variable()` (*owlapy.converter.VariablesMapping method*), 68  
`get_version_iri()` (*owlapy.owl\_ontology.OwlOntologyID method*), 109  
`getOntologyFormat()` (*owlapy.owl\_ontology\_manager.SyncOntologyManager method*), 118  
`GMONTH` (*owlapy.vocab.XSDVocabulary attribute*), 154  
`GMONTHDAY` (*owlapy.vocab.XSDVocabulary attribute*), 154  
`GMonthDayOwlDatatype` (*in module owlapy.owl\_literal*), 103  
`GMonthOwlDatatype` (*in module owlapy.owl\_literal*), 103  
`grammar` (*owlapy.parser.DLSyntaxParser attribute*), 139  
`grammar` (*owlapy.parser.ManchesterOWLSyntaxParser attribute*), 136  
`grouping_vars` (*owlapy.converter.Owl2SparqlConverter attribute*), 69  
`GYEAR` (*owlapy.vocab.XSDVocabulary attribute*), 154  
`GYEARMONTH` (*owlapy.vocab.XSDVocabulary attribute*), 154  
`GYearMonthOwlDatatype` (*in module owlapy.owl\_literal*), 103  
`GYearOwlDatatype` (*in module owlapy.owl\_literal*), 103

## H

`has_consistent_ontology()` (*owlapy.owl\_reasoner.SyncReasoner method*), 133  
`has_float_special_value()` (*owlapy.owl\_literal.OwlLiteral method*), 107



HasCardinality (class in owlapy.meta\_classes), 73  
 HasFiller (class in owlapy.meta\_classes), 72  
 HasIndex (class in owlapy.utils), 150  
 HasIRI (class in owlapy.meta\_classes), 72  
 HasOperands (class in owlapy.meta\_classes), 72  
 having\_conditions (owlapy.converter.Owl2SparqlConverter attribute), 69

## I

import\_and\_include\_axioms\_generators() (in module owlapy.owl\_reasoner), 135  
 ind\_cnt (owlapy.converter.VariablesMapping attribute), 68  
 ind\_data\_properties() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 25  
 ind\_data\_properties() (owlapy.abstracts.AbstractOWLReasoner method), 35  
 ind\_object\_properties() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 25  
 ind\_object\_properties() (owlapy.abstracts.AbstractOWLReasoner method), 35  
 individuals() (owlapy.class\_expression.OwlObjectOneOf method), 67  
 individuals() (owlapy.class\_expression.restriction.OwlObjectOneOf method), 48  
 individuals() (owlapy.owl\_axiom.OwlNaryIndividualAxiom method), 81  
 individuals\_in\_signature() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 16  
 individuals\_in\_signature() (owlapy.abstracts.AbstractOWLOntology method), 27  
 individuals\_in\_signature() (owlapy.owl\_ontology.Ontology method), 110  
 individuals\_in\_signature() (owlapy.owl\_ontology.SyncOntology method), 113  
 infer\_axioms() (owlapy.owl\_reasoner.SyncReasoner method), 133  
 infer\_axioms\_and\_save() (owlapy.owl\_reasoner.SyncReasoner method), 134  
 inference\_types\_mapping (owlapy.owl\_reasoner.SyncReasoner attribute), 128  
 init() (in module owlapy.owlapi\_mapper), 135  
 initialize\_reasoner() (in module owlapy.owl\_reasoner), 135  
 instances() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 22  
 instances() (owlapy.abstracts.AbstractOWLReasoner method), 32  
 instances() (owlapy.owl\_reasoner.StructuralReasoner method), 125  
 instances() (owlapy.owl\_reasoner.SyncReasoner method), 128  
 INTEGER (owlapy.vocab.XSDVocabulary attribute), 153  
 IntegerOWLDatatype (in module owlapy.owl\_literal), 103  
 IRI (class in owlapy.iri), 71  
 iri (owlapy.class\_expression.owl\_class.OwlClass property), 39  
 iri (owlapy.class\_expression.OwlClass property), 56  
 iri (owlapy.meta\_classes.HasIRI property), 72  
 iri (owlapy.owl\_axiom.OwlAnnotationProperty property), 85  
 iri (owlapy.owl\_datatype.OwlDatatype property), 96  
 iri (owlapy.owl\_individual.OwlNamedIndividual property), 100  
 iri (owlapy.owl\_ontology\_manager.OwlImportsDeclaration property), 116  
 iri (owlapy.owl\_property.OwlProperty property), 120  
 is\_annotated() (owlapy.owl\_axiom.OwlAxiom method), 77  
 is\_annotation\_axiom() (owlapy.owl\_axiom.OwlAnnotationAxiom method), 85  
 is\_annotation\_axiom() (owlapy.owl\_axiom.OwlAxiom method), 77  
 is\_anonymous() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 17  
 is\_anonymous() (owlapy.abstracts.AbstractOWLOntology method), 29  
 is\_anonymous() (owlapy.owl\_object.OwlEntity method), 108  
 is\_anonymous() (owlapy.owl\_object.OwlObject method), 107  
 is\_anonymous() (owlapy.owl\_ontology.OwlOntologyID method), 109  
 is\_boolean() (owlapy.owl\_literal.OwlLiteral method), 104  
 is\_child\_of() (owlapy.owl\_hierarchy.AbstractHierarchy method), 97  
 is\_data\_property\_expression() (owlapy.owl\_property.OwlDataPropertyExpression method), 119  
 is\_data\_property\_expression() (owlapy.owl\_property.OwlPropertyExpression method), 118  
 is\_data\_restriction() (owlapy.class\_expression.OwlDataRestriction method), 59  
 is\_data\_restriction() (owlapy.class\_expression.OwlRestriction method), 57  
 is\_data\_restriction() (owlapy.class\_expression.restriction.OwlDataRestriction method), 48  
 is\_data\_restriction() (owlapy.class\_expression.restriction.OwlRestriction method), 43  
 is\_date() (owlapy.owl\_literal.OwlLiteral method), 105  
 is\_datetime() (owlapy.owl\_literal.OwlLiteral method), 105  
 is\_decimal() (owlapy.owl\_literal.OwlLiteral method), 105  
 is\_double() (owlapy.owl\_literal.OwlLiteral method), 104  
 is\_duration() (owlapy.owl\_literal.OwlLiteral method), 105  
 is\_entailed() (owlapy.owl\_reasoner.SyncReasoner method), 134  
 is\_float() (owlapy.owl\_literal.OwlLiteral method), 104  
 is\_gday() (owlapy.owl\_literal.OwlLiteral method), 106  
 is\_gmonth() (owlapy.owl\_literal.OwlLiteral method), 106  
 is\_gmonthday() (owlapy.owl\_literal.OwlLiteral method), 106

- `is_gyear()` (*owlapy.owl\_literal.OWLLiteral method*), 106
- `is_gyearmonth()` (*owlapy.owl\_literal.OWLLiteral method*), 106
- `is_integer()` (*owlapy.owl\_literal.OWLLiteral method*), 105
- `is_literal()` (*owlapy.owl\_annotation.OWLAnnotationValue method*), 75
- `is_literal()` (*owlapy.owl\_literal.OWLLiteral method*), 107
- `is_logical_axiom()` (*owlapy.owl\_axiom.OWLXiom method*), 77
- `is_logical_axiom()` (*owlapy.owl\_axiom.OWLLogicalXiom method*), 77
- `is_modified()` (*owlapy.owl\_ontology.Ontology attribute*), 110
- `is_nothing()` (*owlapy.iri.IRI method*), 71
- `is_object_property_expression()` (*owlapy.owl\_property.OWLObjectPropertyExpression method*), 119
- `is_object_property_expression()` (*owlapy.owl\_property.OWLPropertyExpression method*), 119
- `is_object_restriction()` (*owlapy.class\_expression.OWLObjectRestriction method*), 58
- `is_object_restriction()` (*owlapy.class\_expression.OWLRestriction method*), 57
- `is_object_restriction()` (*owlapy.class\_expression.restriction.OWLObjectRestriction method*), 43
- `is_object_restriction()` (*owlapy.class\_expression.restriction.OWLRestriction method*), 43
- `is_owl_nothing()` (*owlapy.class\_expression.class\_expression.OWLAnonymousClassExpression method*), 37
- `is_owl_nothing()` (*owlapy.class\_expression.class\_expression.OWLClassExpression method*), 36
- `is_owl_nothing()` (*owlapy.class\_expression.owl\_class.OWLClass method*), 40
- `is_owl_nothing()` (*owlapy.class\_expression.OWLAnonymousClassExpression method*), 54
- `is_owl_nothing()` (*owlapy.class\_expression.OWLClass method*), 56
- `is_owl_nothing()` (*owlapy.class\_expression.OWLClassExpression method*), 54
- `is_owl_thing()` (*owlapy.class\_expression.class\_expression.OWLAnonymousClassExpression method*), 37
- `is_owl_thing()` (*owlapy.class\_expression.class\_expression.OWLClassExpression method*), 36
- `is_owl_thing()` (*owlapy.class\_expression.owl\_class.OWLClass method*), 40
- `is_owl_thing()` (*owlapy.class\_expression.OWLAnonymousClassExpression method*), 55
- `is_owl_thing()` (*owlapy.class\_expression.OWLClass method*), 56
- `is_owl_thing()` (*owlapy.class\_expression.OWLClassExpression method*), 54
- `is_owl_top_data_property()` (*owlapy.owl\_property.OWLDataProperty method*), 121
- `is_owl_top_data_property()` (*owlapy.owl\_property.OWLPropertyExpression method*), 119
- `is_owl_top_object_property()` (*owlapy.owl\_property.OWLObjectProperty method*), 120
- `is_owl_top_object_property()` (*owlapy.owl\_property.OWLPropertyExpression method*), 119
- `is_parent_of()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 97
- `is_reserved_vocabulary()` (*owlapy.iri.IRI method*), 71
- `is_satisfiable()` (*owlapy.owl\_reasoner.SyncReasoner method*), 135
- `is_string()` (*owlapy.owl\_literal.OWLLiteral method*), 105
- `is_sub_property_of()` (*owlapy.owl\_hierarchy.DatatypePropertyHierarchy method*), 99
- `is_sub_property_of()` (*owlapy.owl\_hierarchy.ObjectPropertyHierarchy method*), 99
- `is_subclass_of()` (*owlapy.owl\_hierarchy.ClassHierarchy method*), 98
- `is_thing()` (*owlapy.iri.IRI method*), 71
- `is_time()` (*owlapy.owl\_literal.OWLLiteral method*), 106
- `items()` (*owlapy.utils.EvaluatedDescriptionSet attribute*), 150
- `items()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 98
- `iter_count()` (*in module owlapy.utils*), 151

## K

- `KEY` (*owlapy.utils.LRUCache attribute*), 152

## L

- `leaves()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 98
- `LENGTH` (*owlapy.class\_expression.OWLFacet attribute*), 63
- `LENGTH` (*owlapy.vocab.OWLFacet attribute*), 154
- `length()` (*owlapy.utils.OWLClassExpressionLengthMetric method*), 149
- `Literals` (*in module owlapy.class\_expression.restriction*), 43
- `Literals` (*in module owlapy.owl\_literal*), 104
- `load_ontology()` (*owlapy.abstracts.abstract\_owl\_ontology\_manager.AbstractOWLOntologyManager method*), 18
- `load_ontology()` (*owlapy.abstracts.AbstractOWLOntologyManager method*), 26
- `load_ontology()` (*owlapy.OntologyManager method*), 156
- `load_ontology()` (*owlapy.owl\_ontology\_manager.OntologyManager method*), 117
- `load_ontology()` (*owlapy.owl\_ontology\_manager.SyncOntologyManager method*), 117
- `lock` (*owlapy.utils.LRUCache attribute*), 152
- `logger` (*in module owlapy.abstracts.abstract\_owl\_reasoner*), 19
- `logger` (*in module owlapy.owl\_ontology*), 109
- `logger` (*in module owlapy.owl\_reasoner*), 122
- `LONG` (*owlapy.vocab.XSDVocabulary attribute*), 153
- `LRUCache` (*class in owlapy.utils*), 152

## M

manager (*owlapy.owl\_ontology.SyncOntology attribute*), 112  
 MANCHESTER\_GRAMMAR (*in module owlapy.parser*), 136  
 manchester\_to\_owl\_expression() (*in module owlapy*), 155  
 manchester\_to\_owl\_expression() (*in module owlapy.parser*), 141  
 ManchesterOWLSyntaxOWLObjectRenderer (*class in owlapy.render*), 143  
 ManchesterOWLSyntaxParser (*class in owlapy.parser*), 136  
 ManchesterParser (*in module owlapy.parser*), 140  
 ManchesterRenderer (*in module owlapy.render*), 143  
 map\_() (*owlapy.owlapi\_mapper.OWLAPIMapper method*), 135  
 map\_concept() (*owlapy.owl\_ontology.FromOwlready2 method*), 115  
 map\_concept() (*owlapy.owl\_ontology.ToOwlready2 method*), 115  
 map\_datarange() (*owlapy.owl\_ontology.FromOwlready2 method*), 115  
 map\_datarange() (*owlapy.owl\_ontology.ToOwlready2 method*), 115  
 map\_object() (*owlapy.owl\_ontology.ToOwlready2 method*), 115  
 mapper (*in module owlapy.render*), 142  
 mapper (*owlapy.owl\_ontology.SyncOntology attribute*), 112  
 mapper (*owlapy.owl\_reasoner.SyncReasoner attribute*), 128  
 mapping (*owlapy.converter.Owl2SparqlConverter attribute*), 69  
 MAX\_EXCLUSIVE (*owlapy.class\_expression.OWLFacet attribute*), 63  
 MAX\_EXCLUSIVE (*owlapy.vocab.OWLFacet attribute*), 154  
 MAX\_INCLUSIVE (*owlapy.class\_expression.OWLFacet attribute*), 63  
 MAX\_INCLUSIVE (*owlapy.vocab.OWLFacet attribute*), 154  
 MAX\_LENGTH (*owlapy.class\_expression.OWLFacet attribute*), 63  
 MAX\_LENGTH (*owlapy.vocab.OWLFacet attribute*), 154  
 maxsize (*owlapy.utils.LRUCache attribute*), 152  
 maybe\_add() (*owlapy.utils.EvaluatedDescriptionSet method*), 150  
 measurer (*in module owlapy.utils*), 149  
 MIN\_EXCLUSIVE (*owlapy.class\_expression.OWLFacet attribute*), 63  
 MIN\_EXCLUSIVE (*owlapy.vocab.OWLFacet attribute*), 154  
 MIN\_INCLUSIVE (*owlapy.class\_expression.OWLFacet attribute*), 63  
 MIN\_INCLUSIVE (*owlapy.vocab.OWLFacet attribute*), 154  
 MIN\_LENGTH (*owlapy.class\_expression.OWLFacet attribute*), 63  
 MIN\_LENGTH (*owlapy.vocab.OWLFacet attribute*), 154  
 modal\_depth (*owlapy.converter.Owl2SparqlConverter property*), 69  
 module  
     owlapy, 15  
     owlapy.abstracts, 15  
     owlapy.abstracts.abstract\_owl\_ontology, 15  
     owlapy.abstracts.abstract\_owl\_ontology\_manager, 18  
     owlapy.abstracts.abstract\_owl\_reasoner, 19  
     owlapy.class\_expression, 36  
     owlapy.class\_expression.class\_expression, 36  
     owlapy.class\_expression.nary\_boolean\_expression, 38  
     owlapy.class\_expression.owl\_class, 39  
     owlapy.class\_expression.restriction, 40  
     owlapy.converter, 67  
     owlapy.entities, 70  
     owlapy.iri, 70  
     owlapy.meta\_classes, 72  
     owlapy.namespaces, 73  
     owlapy.owl\_annotation, 74  
     owlapy.owl\_axiom, 75  
     owlapy.owl\_data\_ranges, 94  
     owlapy.owl\_datatype, 95  
     owlapy.owl\_hierarchy, 96  
     owlapy.owl\_individual, 100  
     owlapy.owl\_literal, 100  
     owlapy.owl\_object, 107  
     owlapy.owl\_ontology, 108  
     owlapy.owl\_ontology\_manager, 116  
     owlapy.owl\_property, 118  
     owlapy.owl\_reasoner, 122  
     owlapy.owlapi\_mapper, 135  
     owlapy.parser, 136  
     owlapy.providers, 141  
     owlapy.render, 142  
     owlapy.static\_funcs, 144

- owlapy.util\_owl\_static\_funcs, 144
- owlapy.utils, 146
- owlapy.vocab, 152
- more\_general\_roles() (owlapy.owl\_hierarchy.DatatypePropertyHierarchy method), 99
- more\_general\_roles() (owlapy.owl\_hierarchy.ObjectPropertyHierarchy method), 99
- more\_special\_roles() (owlapy.owl\_hierarchy.DatatypePropertyHierarchy method), 99
- more\_special\_roles() (owlapy.owl\_hierarchy.ObjectPropertyHierarchy method), 99
- most\_general\_roles() (owlapy.owl\_hierarchy.DatatypePropertyHierarchy method), 99
- most\_general\_roles() (owlapy.owl\_hierarchy.ObjectPropertyHierarchy method), 99
- most\_special\_roles() (owlapy.owl\_hierarchy.DatatypePropertyHierarchy method), 100
- most\_special\_roles() (owlapy.owl\_hierarchy.ObjectPropertyHierarchy method), 99
- move() (in module owlapy.static\_funcs), 144

## N

- named\_classes() (owlapy.owl\_axiom.OWLEquivalentClassesAxiom method), 81
- named\_individuals (owlapy.converter.Owl2SparqlConverter attribute), 69
- Namespaces (class in owlapy.namespaces), 73
- NAN (owlapy.owl\_literal.FloatSpecialValue attribute), 104
- NEG\_INF (owlapy.owl\_literal.FloatSpecialValue attribute), 104
- NEGATIVEINTEGER (owlapy.vocab.XSDVocabulary attribute), 153
- NegativeIntegerOWLDatatype (in module owlapy.owl\_literal), 103
- new (owlapy.owl\_ontology.SyncOntology attribute), 112
- new\_count\_var() (owlapy.converter.Owl2SparqlConverter method), 69
- new\_individual\_variable() (owlapy.converter.VariablesMapping method), 68
- new\_property\_variable() (owlapy.converter.VariablesMapping method), 68
- NEXT (owlapy.utils.LRUCache attribute), 152
- NNF (class in owlapy.utils), 151
- NONNEGATIVEINTEGER (owlapy.vocab.XSDVocabulary attribute), 153
- NonNegativeIntegerOWLDatatype (in module owlapy.owl\_literal), 103
- NONPOSITIVEINTEGER (owlapy.vocab.XSDVocabulary attribute), 153
- NonPositiveIntegerOWLDatatype (in module owlapy.owl\_literal), 103
- ns (owlapy.namespaces.Namespaces property), 74
- ns (owlapy.parser.DLSyntaxParser attribute), 139
- ns (owlapy.parser.ManchesterOWLSyntaxParser attribute), 136
- NUMERIC\_DATATYPES (in module owlapy.owl\_literal), 103

## O

- o (owlapy.utils.OrderedOWLObject attribute), 150
- object\_all\_values\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_cardinality\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_complement\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_has\_self\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_has\_value\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_intersection\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_inverse\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_one\_of\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_properties\_in\_signature() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 16
- object\_properties\_in\_signature() (owlapy.abstracts.AbstractOWLOntology method), 27
- object\_properties\_in\_signature() (owlapy.owl\_ontology.Ontology method), 110
- object\_properties\_in\_signature() (owlapy.owl\_ontology.SyncOntology method), 113
- object\_property\_domain\_axioms() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 17
- object\_property\_domain\_axioms() (owlapy.abstracts.AbstractOWLOntology method), 28
- object\_property\_domain\_axioms() (owlapy.owl\_ontology.Ontology method), 111
- object\_property\_domain\_axioms() (owlapy.owl\_ontology.SyncOntology method), 114
- object\_property\_domains() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 20
- object\_property\_domains() (owlapy.abstracts.AbstractOWLReasoner method), 29
- object\_property\_domains() (owlapy.owl\_reasoner.StructuralReasoner method), 123
- object\_property\_domains() (owlapy.owl\_reasoner.SyncReasoner method), 130
- object\_property\_length (owlapy.utils.OWLClassExpressionLengthMetric attribute), 149
- object\_property\_range\_axioms() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 17
- object\_property\_range\_axioms() (owlapy.abstracts.AbstractOWLOntology method), 28
- object\_property\_range\_axioms() (owlapy.owl\_ontology.Ontology method), 111
- object\_property\_range\_axioms() (owlapy.owl\_ontology.SyncOntology method), 114
- object\_property\_ranges() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 20
- object\_property\_ranges() (owlapy.abstracts.AbstractOWLReasoner method), 30
- object\_property\_ranges() (owlapy.owl\_reasoner.StructuralReasoner method), 123
- object\_property\_ranges() (owlapy.owl\_reasoner.SyncReasoner method), 130

`object_property_values()` (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method*), 22  
`object_property_values()` (*owlapy.abstracts.AbstractOWLReasoner method*), 32  
`object_property_values()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 125  
`object_property_values()` (*owlapy.owl\_reasoner.SyncReasoner method*), 132  
`object_some_values_length` (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149  
`object_union_length` (*owlapy.utils.OWLClassExpressionLengthMetric attribute*), 149  
`ObjectPropertyHierarchy` (*class in owlapy.owl\_hierarchy*), 98  
`Ontology` (*class in owlapy.owl\_ontology*), 110  
`OntologyManager` (*class in owlapy*), 155  
`OntologyManager` (*class in owlapy.owl\_ontology\_manager*), 116  
`operands()` (*owlapy.class\_expression.class\_expression.OWLObjectComplementOf method*), 38  
`operands()` (*owlapy.class\_expression.nary\_boolean\_expression.OWLNaryBooleanClassExpression method*), 38  
`operands()` (*owlapy.class\_expression.OWLDataOneOf method*), 60  
`operands()` (*owlapy.class\_expression.OWLNaryBooleanClassExpression method*), 57  
`operands()` (*owlapy.class\_expression.OWLObjectComplementOf method*), 55  
`operands()` (*owlapy.class\_expression.OWLObjectOneOf method*), 67  
`operands()` (*owlapy.class\_expression.restriction.OWLDataOneOf method*), 51  
`operands()` (*owlapy.class\_expression.restriction.OWLObjectOneOf method*), 48  
`operands()` (*owlapy.meta\_classes.HasOperands method*), 72  
`operands()` (*owlapy.owl\_axiom.OWLHasKeyAxiom method*), 79  
`operands()` (*owlapy.owl\_data\_ranges.OWLNaryDataRange method*), 94  
`OperandSetTransform` (*class in owlapy.utils*), 150  
`operator` (*owlapy.class\_expression.OWLFacet property*), 63  
`operator` (*owlapy.vocab.OWLFacet property*), 154  
`OrderedOWLObject` (*class in owlapy.utils*), 150  
`OWL` (*in module owlapy.namespaces*), 74  
`owl2sparqlConverter` (*class in owlapy.converter*), 68  
`OWL_BOTTOM_DATA_PROPERTY` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWL_BOTTOM_OBJECT_PROPERTY` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWL_CLASS` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`owl_datatype_max_exclusive_restriction()` (*in module owlapy.providers*), 141  
`owl_datatype_max_inclusive_restriction()` (*in module owlapy.providers*), 141  
`owl_datatype_min_exclusive_restriction()` (*in module owlapy.providers*), 141  
`owl_datatype_min_inclusive_restriction()` (*in module owlapy.providers*), 141  
`owl_datatype_min_max_exclusive_restriction()` (*in module owlapy.providers*), 141  
`owl_datatype_min_max_inclusive_restriction()` (*in module owlapy.providers*), 141  
`owl_expression_to_dl()` (*in module owlapy*), 155  
`owl_expression_to_dl()` (*in module owlapy.render*), 144  
`owl_expression_to_manchester()` (*in module owlapy*), 155  
`owl_expression_to_manchester()` (*in module owlapy.render*), 144  
`owl_expression_to_sparql()` (*in module owlapy*), 155  
`owl_expression_to_sparql()` (*in module owlapy.converter*), 70  
`owl_expression_to_sparql_with_confusion_matrix()` (*in module owlapy*), 155  
`owl_expression_to_sparql_with_confusion_matrix()` (*in module owlapy.converter*), 70  
`OWL_NAMED_INDIVIDUAL` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWL_NOTHING` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWL_THING` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWL_TOP_DATA_PROPERTY` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWL_TOP_OBJECT_PROPERTY` (*owlapy.vocab.OWLRDFVocabulary attribute*), 153  
`OWLAnnotation` (*class in owlapy.owl\_axiom*), 85  
`OWLAnnotationAssertionAxiom` (*class in owlapy.owl\_axiom*), 85  
`OWLAnnotationAxiom` (*class in owlapy.owl\_axiom*), 85  
`OWLAnnotationObject` (*class in owlapy.owl\_annotation*), 74  
`OWLAnnotationProperty` (*class in owlapy.owl\_axiom*), 85  
`OWLAnnotationPropertyDomainAxiom` (*class in owlapy.owl\_axiom*), 86  
`OWLAnnotationPropertyRangeAxiom` (*class in owlapy.owl\_axiom*), 87  
`OWLAnnotationSubject` (*class in owlapy.owl\_annotation*), 74  
`OWLAnnotationValue` (*class in owlapy.owl\_annotation*), 74  
`OWLANonymousClassExpression` (*class in owlapy.class\_expression*), 54  
`OWLANonymousClassExpression` (*class in owlapy.class\_expression.class\_expression*), 37  
`owlapi_manager` (*owlapy.owl\_ontology\_manager.SyncOntologyManager attribute*), 117  
`OWLAPIMapper` (*class in owlapy.owlapi\_mapper*), 135  
`owlapy`  
    *module*, 15  
`owlapy.abstracts`  
    *module*, 15  
`owlapy.abstracts.abstract_owl_ontology`  
    *module*, 15

- owlapy.abstracts.abstract\_owl\_ontology\_manager  
module, 18
- owlapy.abstracts.abstract\_owl\_reasoner  
module, 19
- owlapy.class\_expression  
module, 36
- owlapy.class\_expression.class\_expression  
module, 36
- owlapy.class\_expression.nary\_boolean\_expression  
module, 38
- owlapy.class\_expression.owl\_class  
module, 39
- owlapy.class\_expression.restriction  
module, 40
- owlapy.converter  
module, 67
- owlapy.entities  
module, 70
- owlapy.iri  
module, 70
- owlapy.meta\_classes  
module, 72
- owlapy.namespaces  
module, 73
- owlapy.owl\_annotation  
module, 74
- owlapy.owl\_axiom  
module, 75
- owlapy.owl\_data\_ranges  
module, 94
- owlapy.owl\_datatype  
module, 95
- owlapy.owl\_hierarchy  
module, 96
- owlapy.owl\_individual  
module, 100
- owlapy.owl\_literal  
module, 100
- owlapy.owl\_object  
module, 107
- owlapy.owl\_ontology  
module, 108
- owlapy.owl\_ontology\_manager  
module, 116
- owlapy.owl\_property  
module, 118
- owlapy.owl\_reasoner  
module, 122
- owlapy.owlapi\_mapper  
module, 135
- owlapy.parser  
module, 136
- owlapy.providers  
module, 141
- owlapy.render  
module, 142
- owlapy.static\_funcs  
module, 144
- owlapy.util\_owl\_static\_funcs  
module, 144
- owlapy.utils  
module, 146
- owlapy.vocab  
module, 152
- OWLAsymmetricObjectPropertyAxiom (*class in owlapy.owl\_axiom*), 90
- OWLAxiom (*class in owlapy.owl\_axiom*), 77
- OWLBooleanClassExpression (*class in owlapy.class\_expression*), 55
- OWLBooleanClassExpression (*class in owlapy.class\_expression.class\_expression*), 37



OWLBottomDataProperty (in module owlapy.owl\_literal), 103  
 OWLBottomObjectProperty (in module owlapy.owl\_literal), 103  
 OWLCardinalityRestriction (class in owlapy.class\_expression), 59  
 OWLCardinalityRestriction (class in owlapy.class\_expression.restriction), 44  
 OWLClass (class in owlapy.class\_expression), 55  
 OWLClass (class in owlapy.class\_expression.owl\_class), 39  
 OWLClassAssertionAxiom (class in owlapy.owl\_axiom), 84  
 OWLClassAxiom (class in owlapy.owl\_axiom), 78  
 OWLClassExpression (class in owlapy.class\_expression), 54  
 OWLClassExpression (class in owlapy.class\_expression.class\_expression), 36  
 OWLClassExpressionLengthMetric (class in owlapy.utils), 148  
 OWLDataAllValuesFrom (class in owlapy.class\_expression), 65  
 OWLDataAllValuesFrom (class in owlapy.class\_expression.restriction), 50  
 OWLDataCardinalityRestriction (class in owlapy.class\_expression), 61  
 OWLDataCardinalityRestriction (class in owlapy.class\_expression.restriction), 48  
 OWLDataComplementOf (class in owlapy.owl\_data\_ranges), 95  
 OWLDataExactCardinality (class in owlapy.class\_expression), 66  
 OWLDataExactCardinality (class in owlapy.class\_expression.restriction), 49  
 OWLDataHasValue (class in owlapy.class\_expression), 65  
 OWLDataHasValue (class in owlapy.class\_expression.restriction), 51  
 OWLDataIntersectionOf (class in owlapy.owl\_data\_ranges), 94  
 OWLDataMaxCardinality (class in owlapy.class\_expression), 66  
 OWLDataMaxCardinality (class in owlapy.class\_expression.restriction), 49  
 OWLDataMinCardinality (class in owlapy.class\_expression), 66  
 OWLDataMinCardinality (class in owlapy.class\_expression.restriction), 49  
 OWLDataOneOf (class in owlapy.class\_expression), 60  
 OWLDataOneOf (class in owlapy.class\_expression.restriction), 51  
 OWLDataProperty (class in owlapy.owl\_property), 121  
 OWLDataPropertyAssertionAxiom (class in owlapy.owl\_axiom), 89  
 OWLDataPropertyAxiom (class in owlapy.owl\_axiom), 78  
 OWLDataPropertyCharacteristicAxiom (class in owlapy.owl\_axiom), 91  
 OWLDataPropertyDomainAxiom (class in owlapy.owl\_axiom), 93  
 OWLDataPropertyExpression (class in owlapy.owl\_property), 119  
 OWLDataPropertyRangeAxiom (class in owlapy.owl\_axiom), 93  
 OWLDataRange (class in owlapy.owl\_data\_ranges), 94  
 OWLDataRestriction (class in owlapy.class\_expression), 59  
 OWLDataRestriction (class in owlapy.class\_expression.restriction), 48  
 OWLDataSomeValuesFrom (class in owlapy.class\_expression), 64  
 OWLDataSomeValuesFrom (class in owlapy.class\_expression.restriction), 50  
 OWLDatatype (class in owlapy.owl\_datatype), 96  
 OWLDatatypeDefinitionAxiom (class in owlapy.owl\_axiom), 78  
 OWLDatatypeRestriction (class in owlapy.class\_expression), 62  
 OWLDatatypeRestriction (class in owlapy.class\_expression.restriction), 52  
 OWLDataUnionOf (class in owlapy.owl\_data\_ranges), 95  
 OWLDeclarationAxiom (class in owlapy.owl\_axiom), 78  
 OWLDifferentIndividualsAxiom (class in owlapy.owl\_axiom), 81  
 OWLDisjointClassesAxiom (class in owlapy.owl\_axiom), 81  
 OWLDisjointDataPropertiesAxiom (class in owlapy.owl\_axiom), 83  
 OWLDisjointObjectPropertiesAxiom (class in owlapy.owl\_axiom), 82  
 OWLDisjointUnionAxiom (class in owlapy.owl\_axiom), 84  
 OWLEntity (class in owlapy.owl\_object), 108  
 OWLEquivalentClassesAxiom (class in owlapy.owl\_axiom), 80  
 OWLEquivalentDataPropertiesAxiom (class in owlapy.owl\_axiom), 83  
 OWLEquivalentObjectPropertiesAxiom (class in owlapy.owl\_axiom), 82  
 OWLFacet (class in owlapy.class\_expression), 63  
 OWLFacet (class in owlapy.vocab), 154  
 OWLFacetRestriction (class in owlapy.class\_expression), 63  
 OWLFacetRestriction (class in owlapy.class\_expression.restriction), 52  
 OWLFunctionalDataPropertyAxiom (class in owlapy.owl\_axiom), 92  
 OWLFunctionalObjectPropertyAxiom (class in owlapy.owl\_axiom), 90  
 OWLHasKeyAxiom (class in owlapy.owl\_axiom), 79  
 OWLHasValueRestriction (class in owlapy.class\_expression), 58  
 OWLHasValueRestriction (class in owlapy.class\_expression.restriction), 43  
 OWLImportsDeclaration (class in owlapy.owl\_ontology\_manager), 116  
 OWLIndividual (class in owlapy.owl\_individual), 100  
 OWLIndividualAxiom (class in owlapy.owl\_axiom), 78  
 OWLInverseFunctionalObjectPropertyAxiom (class in owlapy.owl\_axiom), 90  
 OWLInverseObjectPropertiesAxiom (class in owlapy.owl\_axiom), 82

OWLIrreflexiveObjectPropertyAxiom (class in owlapy.owl\_axiom), 90  
 OWLLiteral (class in owlapy.owl\_literal), 104  
 OWLLogicalAxiom (class in owlapy.owl\_axiom), 77  
 OWLNamedIndividual (class in owlapy.owl\_individual), 100  
 OWLNamedObject (class in owlapy.owl\_object), 108  
 OWLNaryAxiom (class in owlapy.owl\_axiom), 79  
 OWLNaryBooleanClassExpression (class in owlapy.class\_expression), 56  
 OWLNaryBooleanClassExpression (class in owlapy.class\_expression.nary\_boolean\_expression), 38  
 OWLNaryClassAxiom (class in owlapy.owl\_axiom), 80  
 OWLNaryDataRange (class in owlapy.owl\_data\_ranges), 94  
 OWLNaryIndividualAxiom (class in owlapy.owl\_axiom), 81  
 OWLNaryPropertyAxiom (class in owlapy.owl\_axiom), 82  
 OWLNegativeDataPropertyAssertionAxiom (class in owlapy.owl\_axiom), 89  
 OWLNegativeObjectPropertyAssertionAxiom (class in owlapy.owl\_axiom), 89  
 OWLObject (class in owlapy.owl\_object), 107  
 OWLObjectAllValuesFrom (class in owlapy.class\_expression), 61  
 OWLObjectAllValuesFrom (class in owlapy.class\_expression.restriction), 46  
 OWLObjectCardinalityRestriction (class in owlapy.class\_expression), 59  
 OWLObjectCardinalityRestriction (class in owlapy.class\_expression.restriction), 44  
 OWLObjectComplementOf (class in owlapy.class\_expression), 55  
 OWLObjectComplementOf (class in owlapy.class\_expression.class\_expression), 37  
 OWLObjectExactCardinality (class in owlapy.class\_expression), 64  
 OWLObjectExactCardinality (class in owlapy.class\_expression.restriction), 45  
 OWLObjectHasSelf (class in owlapy.class\_expression), 60  
 OWLObjectHasSelf (class in owlapy.class\_expression.restriction), 46  
 OWLObjectHasValue (class in owlapy.class\_expression), 62  
 OWLObjectHasValue (class in owlapy.class\_expression.restriction), 47  
 OWLObjectIntersectionOf (class in owlapy.class\_expression), 57  
 OWLObjectIntersectionOf (class in owlapy.class\_expression.nary\_boolean\_expression), 39  
 OWLObjectInverseOf (class in owlapy.owl\_property), 120  
 OWLObjectMaxCardinality (class in owlapy.class\_expression), 64  
 OWLObjectMaxCardinality (class in owlapy.class\_expression.restriction), 45  
 OWLObjectMinCardinality (class in owlapy.class\_expression), 64  
 OWLObjectMinCardinality (class in owlapy.class\_expression.restriction), 45  
 OWLObjectOneOf (class in owlapy.class\_expression), 67  
 OWLObjectOneOf (class in owlapy.class\_expression.restriction), 47  
 OWLObjectParser (class in owlapy.owl\_object), 108  
 OWLObjectProperty (class in owlapy.owl\_property), 120  
 OWLObjectPropertyAssertionAxiom (class in owlapy.owl\_axiom), 88  
 OWLObjectPropertyAxiom (class in owlapy.owl\_axiom), 78  
 OWLObjectPropertyCharacteristicAxiom (class in owlapy.owl\_axiom), 90  
 OWLObjectPropertyDomainAxiom (class in owlapy.owl\_axiom), 92  
 OWLObjectPropertyExpression (class in owlapy.owl\_property), 119  
 OWLObjectPropertyRangeAxiom (class in owlapy.owl\_axiom), 93  
 OWLObjectRenderer (class in owlapy.owl\_object), 107  
 OWLObjectRestriction (class in owlapy.class\_expression), 58  
 OWLObjectRestriction (class in owlapy.class\_expression.restriction), 43  
 OWLObjectSomeValuesFrom (class in owlapy.class\_expression), 61  
 OWLObjectSomeValuesFrom (class in owlapy.class\_expression.restriction), 46  
 OWLObjectUnionOf (class in owlapy.class\_expression), 57  
 OWLObjectUnionOf (class in owlapy.class\_expression.nary\_boolean\_expression), 39  
 OWLOntologyID (class in owlapy.owl\_ontology), 109  
 OWLProperty (class in owlapy.owl\_property), 120  
 OWLPropertyAssertionAxiom (class in owlapy.owl\_axiom), 88  
 OWLPropertyAxiom (class in owlapy.owl\_axiom), 78  
 OWLPropertyDomainAxiom (class in owlapy.owl\_axiom), 92  
 OWLPropertyExpression (class in owlapy.owl\_property), 118  
 OWLPropertyRange (class in owlapy.owl\_data\_ranges), 94  
 OWLPropertyRangeAxiom (class in owlapy.owl\_axiom), 92  
 OWLQuantifiedDataRestriction (class in owlapy.class\_expression), 60  
 OWLQuantifiedDataRestriction (class in owlapy.class\_expression.restriction), 48  
 OWLQuantifiedObjectRestriction (class in owlapy.class\_expression), 58  
 OWLQuantifiedObjectRestriction (class in owlapy.class\_expression.restriction), 44  
 OWLQuantifiedRestriction (class in owlapy.class\_expression), 57  
 OWLQuantifiedRestriction (class in owlapy.class\_expression.restriction), 44  
 OWLRDFVocabulary (class in owlapy.vocab), 153  
 OWLREADY2\_FACET\_KEYS (in module owlapy.owl\_ontology), 115  
 OWLReflexiveObjectPropertyAxiom (class in owlapy.owl\_axiom), 91



OWLRestriction (class in owlapy.class\_expression), 57  
 OWLRestriction (class in owlapy.class\_expression.restriction), 43  
 OWLSameIndividualAxiom (class in owlapy.owl\_axiom), 81  
 OWLSubAnnotationPropertyOfAxiom (class in owlapy.owl\_axiom), 86  
 OWLSubClassOfAxiom (class in owlapy.owl\_axiom), 83  
 OWLSubDataPropertyOfAxiom (class in owlapy.owl\_axiom), 88  
 OWLSubObjectPropertyOfAxiom (class in owlapy.owl\_axiom), 87  
 OWLSubPropertyAxiom (class in owlapy.owl\_axiom), 87  
 OWLSymmetricObjectPropertyAxiom (class in owlapy.owl\_axiom), 91  
 OWLTopDataProperty (in module owlapy.owl\_literal), 103  
 OWLTopObjectProperty (in module owlapy.owl\_literal), 103  
 OWLTransitiveObjectPropertyAxiom (class in owlapy.owl\_axiom), 91  
 OWLUnaryPropertyAxiom (class in owlapy.owl\_axiom), 89

## P

parent (owlapy.converter.Owl2SparqlConverter attribute), 69  
 parent\_var (owlapy.converter.Owl2SparqlConverter attribute), 69  
 parents() (owlapy.owl\_hierarchy.AbstractHierarchy method), 97  
 parse\_boolean() (owlapy.owl\_literal.OwLLiteral method), 104  
 parse\_date() (owlapy.owl\_literal.OwLLiteral method), 105  
 parse\_datetime() (owlapy.owl\_literal.OwLLiteral method), 105  
 parse\_decimal() (owlapy.owl\_literal.OwLLiteral method), 105  
 parse\_double() (owlapy.owl\_literal.OwLLiteral method), 104  
 parse\_duration() (owlapy.owl\_literal.OwLLiteral method), 105  
 parse\_expression() (owlapy.owl\_object.OwLObjectParser method), 108  
 parse\_expression() (owlapy.parser.DLSyntaxParser method), 139  
 parse\_expression() (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 parse\_float() (owlapy.owl\_literal.OwLLiteral method), 104  
 parse\_gday() (owlapy.owl\_literal.OwLLiteral method), 106  
 parse\_gmonth() (owlapy.owl\_literal.OwLLiteral method), 106  
 parse\_gmonthday() (owlapy.owl\_literal.OwLLiteral method), 106  
 parse\_gyear() (owlapy.owl\_literal.OwLLiteral method), 106  
 parse\_gyearmonth() (owlapy.owl\_literal.OwLLiteral method), 106  
 parse\_integer() (owlapy.owl\_literal.OwLLiteral method), 105  
 parse\_string() (owlapy.owl\_literal.OwLLiteral method), 105  
 parse\_time() (owlapy.owl\_literal.OwLLiteral method), 106  
 path (owlapy.owl\_ontology.SyncOntology attribute), 112  
 PATTERN (owlapy.class\_expression.OwLFacet attribute), 63  
 PATTERN (owlapy.vocab.OwLFacet attribute), 154  
 peek() (in module owlapy.converter), 68  
 POS\_INF (owlapy.owl\_literal.FloatSpecialValue attribute), 104  
 POSITIVEINTEGER (owlapy.vocab.XSDVocabulary attribute), 153  
 PositiveIntegerOWLDatatype (in module owlapy.owl\_literal), 103  
 prefix (owlapy.namespaces.Namespaces property), 74  
 PREV (owlapy.utils.LRUCache attribute), 152  
 process() (owlapy.converter.Owl2SparqlConverter method), 69  
 prop (owlapy.owl\_axiom.OwLObjectPropertyDomainAxiom property), 93  
 prop (owlapy.owl\_axiom.OwLPropertyRangeAxiom property), 92  
 prop\_cnt (owlapy.converter.VariablesMapping attribute), 68  
 properties (owlapy.converter.Owl2SparqlConverter attribute), 69  
 properties() (owlapy.owl\_axiom.OwLNaryPropertyAxiom method), 82  
 properties\_in\_signature() (owlapy.owl\_ontology.Ontology method), 110

## R

range (owlapy.owl\_axiom.OwLPropertyRangeAxiom property), 92  
 RDF (in module owlapy.namespaces), 74  
 RDFS (in module owlapy.namespaces), 74  
 RDFS\_LITERAL (owlapy.vocab.OwLRDFVocabulary attribute), 153  
 reminder (owlapy.class\_expression.owl\_class.OwLClass property), 40  
 reminder (owlapy.class\_expression.OwLClass property), 56  
 reminder (owlapy.iri.IRI property), 71  
 reminder (owlapy.owl\_individual.OwLNamedIndividual property), 100  
 reminder (owlapy.owl\_property.OwLObjectProperty property), 120  
 remove\_axiom() (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method), 17  
 remove\_axiom() (owlapy.abstracts.AbstractOWLOntology method), 29  
 remove\_axiom() (owlapy.owl\_ontology.Ontology method), 112  
 remove\_axiom() (owlapy.owl\_ontology.SyncOntology method), 115

`render()` (*owlapy.converter.Owl2SparqlConverter method*), 69  
`render()` (*owlapy.owl\_object.OwlObjectRenderer method*), 108  
`render()` (*owlapy.render.DLSyntaxObjectRenderer method*), 143  
`render()` (*owlapy.render.ManchesterOWLSyntaxOwlObjectRenderer method*), 143  
`reset()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 122  
`reset_and_disable_cache()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 128  
`restrict()` (*owlapy.owl\_hierarchy.AbstractHierarchy static method*), 97  
`restrict_and_copy()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 97  
`Restriction_Literals` (in module *owlapy.providers*), 141  
`RESULT` (*owlapy.utils.LRUCache attribute*), 152  
`root` (*owlapy.utils.LRUCache attribute*), 152  
`roots()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 98  
`run_with_timeout()` (in module *owlapy.utils*), 147

## S

`same_individuals()` (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method*), 21  
`same_individuals()` (*owlapy.abstracts.AbstractOWLReasoner method*), 31  
`same_individuals()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 124  
`same_individuals()` (*owlapy.owl\_reasoner.SyncReasoner method*), 131  
`save()` (*owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology method*), 17  
`save()` (*owlapy.abstracts.AbstractOWLOntology method*), 29  
`save()` (*owlapy.owl\_ontology.Ontology method*), 112  
`save()` (*owlapy.owl\_ontology.SyncOntology method*), 115  
`save_owl_class_expressions()` (in module *owlapy.util\_owl\_static\_funcs*), 145  
`save_world()` (*owlapy.OntologyManager method*), 156  
`save_world()` (*owlapy.owl\_ontology\_manager.OntologyManager method*), 117  
`saveOntology()` (*owlapy.owl\_ontology\_manager.SyncOntologyManager method*), 118  
`sentinel` (*owlapy.utils.LRUCache attribute*), 152  
`set_short_form_provider()` (*owlapy.owl\_object.OwlObjectRenderer method*), 107  
`set_short_form_provider()` (*owlapy.render.DLSyntaxObjectRenderer method*), 143  
`set_short_form_provider()` (*owlapy.render.ManchesterOWLSyntaxOwlObjectRenderer method*), 143  
`siblings()` (*owlapy.owl\_hierarchy.AbstractHierarchy method*), 98  
`simplify()` (*owlapy.utils.OperandSetTransform method*), 150  
`slots` (*owlapy.parser.DLSyntaxParser attribute*), 138  
`slots` (*owlapy.parser.ManchesterOWLSyntaxParser attribute*), 136  
`sort()` (*owlapy.utils.ConceptOperandSorter method*), 150  
`sparql` (*owlapy.converter.Owl2SparqlConverter attribute*), 68  
`stack_parent()` (*owlapy.converter.Owl2SparqlConverter method*), 69  
`stack_variable()` (*owlapy.converter.Owl2SparqlConverter method*), 69  
`startJVM()` (in module *owlapy.static\_funcs*), 144  
`stopJVM()` (in module *owlapy.static\_funcs*), 144  
`str` (*owlapy.class\_expression.owl\_class.OWLClass property*), 39  
`str` (*owlapy.class\_expression.OWLClass property*), 56  
`str` (*owlapy.iri.IRI property*), 71  
`str` (*owlapy.meta\_classes.HasIRI property*), 72  
`str` (*owlapy.owl\_axiom.OWLAnnotationProperty property*), 85  
`str` (*owlapy.owl\_datatype.OWLDatatype property*), 96  
`str` (*owlapy.owl\_individual.OWLNamedIndividual property*), 100  
`str` (*owlapy.owl\_ontology\_manager.OWLImportsDeclaration property*), 116  
`str` (*owlapy.owl\_property.OWLProperty property*), 120  
`STRING` (*owlapy.vocab.XSDVocabulary attribute*), 153  
`StringOWLDatatype` (in module *owlapy.owl\_literal*), 103  
`StructuralReasoner` (class in *owlapy.owl\_reasoner*), 122  
`sub_class` (*owlapy.owl\_axiom.OWLSubClassOfAxiom property*), 84  
`sub_classes()` (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method*), 22  
`sub_classes()` (*owlapy.abstracts.AbstractOWLReasoner method*), 32  
`sub_classes()` (*owlapy.owl\_hierarchy.ClassHierarchy method*), 98  
`sub_classes()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 125  
`sub_classes()` (*owlapy.owl\_reasoner.SyncReasoner method*), 129  
`sub_data_properties()` (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method*), 23  
`sub_data_properties()` (*owlapy.abstracts.AbstractOWLReasoner method*), 33  
`sub_data_properties()` (*owlapy.owl\_hierarchy.DatatypePropertyHierarchy method*), 99  
`sub_data_properties()` (*owlapy.owl\_reasoner.StructuralReasoner method*), 127  
`sub_data_properties()` (*owlapy.owl\_reasoner.SyncReasoner method*), 131  
`sub_object_properties()` (*owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method*), 24  
`sub_object_properties()` (*owlapy.abstracts.AbstractOWLReasoner method*), 33  
`sub_object_properties()` (*owlapy.owl\_hierarchy.ObjectPropertyHierarchy method*), 99

sub\_object\_properties() (owlapy.owl\_reasoner.StructuralReasoner method), 127  
 sub\_object\_properties() (owlapy.owl\_reasoner.SyncReasoner method), 130  
 super\_class (owlapy.owl\_axiom.OWLSubClassOfAxiom property), 84  
 super\_classes() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 24  
 super\_classes() (owlapy.abstracts.AbstractOWLReasoner method), 34  
 super\_classes() (owlapy.owl\_hierarchy.ClassHierarchy method), 98  
 super\_classes() (owlapy.owl\_reasoner.StructuralReasoner method), 125  
 super\_classes() (owlapy.owl\_reasoner.SyncReasoner method), 129  
 super\_data\_properties() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 23  
 super\_data\_properties() (owlapy.abstracts.AbstractOWLReasoner method), 33  
 super\_data\_properties() (owlapy.owl\_hierarchy.DatatypePropertyHierarchy method), 99  
 super\_data\_properties() (owlapy.owl\_reasoner.StructuralReasoner method), 126  
 super\_data\_properties() (owlapy.owl\_reasoner.SyncReasoner method), 131  
 super\_object\_properties() (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 24  
 super\_object\_properties() (owlapy.abstracts.AbstractOWLReasoner method), 34  
 super\_object\_properties() (owlapy.owl\_hierarchy.ObjectPropertyHierarchy method), 99  
 super\_object\_properties() (owlapy.owl\_reasoner.StructuralReasoner method), 127  
 super\_object\_properties() (owlapy.owl\_reasoner.SyncReasoner method), 131  
 symbolic\_form (owlapy.class\_expression.OWLFacet property), 63  
 symbolic\_form (owlapy.vocab.OWLFacet property), 154  
 SyncOntology (class in owlapy.owl\_ontology), 112  
 SyncOntologyManager (class in owlapy.owl\_ontology\_manager), 117  
 SyncReasoner (class in owlapy.owl\_reasoner), 128

## T

TIME (owlapy.vocab.XSDVocabulary attribute), 154  
 TIME\_DATATYPES (in module owlapy.owl\_literal), 103  
 TimeOWLDatatype (in module owlapy.owl\_literal), 103  
 to\_list() (owlapy.owlapi\_mapper.OWLAPIMapper static method), 136  
 to\_python() (owlapy.owl\_literal.OWLLiteral method), 107  
 to\_string\_id() (owlapy.owl\_object.OWLEntity method), 108  
 ToOwlready2 (class in owlapy.owl\_ontology), 115  
 TopLevelCNF (class in owlapy.utils), 151  
 TopLevelDNF (class in owlapy.utils), 151  
 TopOWLDatatype (in module owlapy.owl\_literal), 103  
 TOTAL\_DIGITS (owlapy.class\_expression.OWLFacet attribute), 63  
 TOTAL\_DIGITS (owlapy.vocab.OWLFacet attribute), 154  
 translating\_short\_form\_endpoint() (in module owlapy.render), 142  
 translating\_short\_form\_provider() (in module owlapy.render), 142  
 triple() (owlapy.converter.Owl2SparqlConverter method), 70  
 type\_index (owlapy.abstracts.abstract\_owl\_ontology.AbstractOWLOntology attribute), 15  
 type\_index (owlapy.abstracts.AbstractOWLOntology attribute), 27  
 type\_index (owlapy.class\_expression.class\_expression.OWLObjectComplementOf attribute), 38  
 type\_index (owlapy.class\_expression.nary\_boolean\_expression.OWLObjectIntersectionOf attribute), 39  
 type\_index (owlapy.class\_expression.nary\_boolean\_expression.OWLObjectUnionOf attribute), 39  
 type\_index (owlapy.class\_expression.owl\_class.OWLClass attribute), 39  
 type\_index (owlapy.class\_expression.OWLClass attribute), 56  
 type\_index (owlapy.class\_expression.OWLDataAllValuesFrom attribute), 65  
 type\_index (owlapy.class\_expression.OWLDataExactCardinality attribute), 67  
 type\_index (owlapy.class\_expression.OWLDataHasValue attribute), 66  
 type\_index (owlapy.class\_expression.OWLDataMaxCardinality attribute), 66  
 type\_index (owlapy.class\_expression.OWLDataMinCardinality attribute), 66  
 type\_index (owlapy.class\_expression.OWLDataOneOf attribute), 60  
 type\_index (owlapy.class\_expression.OWLDataSomeValuesFrom attribute), 65  
 type\_index (owlapy.class\_expression.OWLDatatypeRestriction attribute), 63  
 type\_index (owlapy.class\_expression.OWLFacetRestriction attribute), 63  
 type\_index (owlapy.class\_expression.OWLObjectAllValuesFrom attribute), 62  
 type\_index (owlapy.class\_expression.OWLObjectComplementOf attribute), 55  
 type\_index (owlapy.class\_expression.OWLObjectExactCardinality attribute), 64  
 type\_index (owlapy.class\_expression.OWLObjectHasSelf attribute), 60  
 type\_index (owlapy.class\_expression.OWLObjectHasValue attribute), 62  
 type\_index (owlapy.class\_expression.OWLObjectIntersectionOf attribute), 57  
 type\_index (owlapy.class\_expression.OWLObjectMaxCardinality attribute), 64  
 type\_index (owlapy.class\_expression.OWLObjectMinCardinality attribute), 64  
 type\_index (owlapy.class\_expression.OWLObjectOneOf attribute), 67  
 type\_index (owlapy.class\_expression.OWLObjectSomeValuesFrom attribute), 61  
 type\_index (owlapy.class\_expression.OWLObjectUnionOf attribute), 57

type\_index (owlapy.class\_expression.restriction.OWLObjectAllValuesFrom attribute), 50  
 type\_index (owlapy.class\_expression.restriction.OWLObjectExactCardinality attribute), 49  
 type\_index (owlapy.class\_expression.restriction.OWLObjectHasValue attribute), 51  
 type\_index (owlapy.class\_expression.restriction.OWLObjectMaxCardinality attribute), 49  
 type\_index (owlapy.class\_expression.restriction.OWLObjectMinCardinality attribute), 49  
 type\_index (owlapy.class\_expression.restriction.OWLObjectOneOf attribute), 51  
 type\_index (owlapy.class\_expression.restriction.OWLObjectSomeValuesFrom attribute), 50  
 type\_index (owlapy.class\_expression.restriction.OWLObjectDatatypeRestriction attribute), 52  
 type\_index (owlapy.class\_expression.restriction.OWLObjectFacetRestriction attribute), 52  
 type\_index (owlapy.class\_expression.restriction.OWLObjectAllValuesFrom attribute), 46  
 type\_index (owlapy.class\_expression.restriction.OWLObjectExactCardinality attribute), 46  
 type\_index (owlapy.class\_expression.restriction.OWLObjectHasSelf attribute), 47  
 type\_index (owlapy.class\_expression.restriction.OWLObjectHasValue attribute), 47  
 type\_index (owlapy.class\_expression.restriction.OWLObjectMaxCardinality attribute), 45  
 type\_index (owlapy.class\_expression.restriction.OWLObjectMinCardinality attribute), 45  
 type\_index (owlapy.class\_expression.restriction.OWLObjectOneOf attribute), 47  
 type\_index (owlapy.class\_expression.restriction.OWLObjectSomeValuesFrom attribute), 46  
 type\_index (owlapy.iri.iri attribute), 71  
 type\_index (owlapy.owl\_data\_ranges.OWLObjectComplementOf attribute), 95  
 type\_index (owlapy.owl\_data\_ranges.OWLObjectIntersectionOf attribute), 95  
 type\_index (owlapy.owl\_data\_ranges.OWLObjectUnionOf attribute), 95  
 type\_index (owlapy.owl\_datatype.OWLObjectDatatype attribute), 96  
 type\_index (owlapy.owl\_individual.OWLObjectNamedIndividual attribute), 100  
 type\_index (owlapy.owl\_literal.OWLObjectLiteral attribute), 104  
 type\_index (owlapy.owl\_property.OWLObjectDataProperty attribute), 121  
 type\_index (owlapy.owl\_property.OWLObjectObjectInverseOf attribute), 121  
 type\_index (owlapy.owl\_property.OWLObjectObjectProperty attribute), 120  
 type\_index (owlapy.utils.HasIndex attribute), 150  
 types () (owlapy.abstracts.abstract\_owl\_reasoner.AbstractOWLReasoner method), 24  
 types () (owlapy.abstracts.AbstractOWLReasoner method), 34  
 types () (owlapy.owl\_reasoner.StructuralReasoner method), 128  
 types () (owlapy.owl\_reasoner.SyncReasoner method), 133

## U

unsatisfiable\_classes () (owlapy.owl\_reasoner.SyncReasoner method), 135

## V

values () (owlapy.class\_expression.OWLObjectOneOf method), 60  
 values () (owlapy.class\_expression.restriction.OWLObjectOneOf method), 51  
 variable\_entities (owlapy.converter.Owl2SparqlConverter attribute), 69  
 variables (owlapy.converter.Owl2SparqlConverter attribute), 69  
 VariablesMapping (class in owlapy.converter), 68  
 visit\_abbreviated\_iri () (owlapy.parser.DLSyntaxParser method), 140  
 visit\_abbreviated\_iri () (owlapy.parser.ManchesterOWLSyntaxParser method), 138  
 visit\_boolean\_literal () (owlapy.parser.DLSyntaxParser method), 140  
 visit\_boolean\_literal () (owlapy.parser.ManchesterOWLSyntaxParser method), 138  
 visit\_cardinality\_res () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_cardinality\_res () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_class\_expression () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_class\_expression () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_class\_iri () (owlapy.parser.DLSyntaxParser method), 140  
 visit\_class\_iri () (owlapy.parser.ManchesterOWLSyntaxParser method), 138  
 visit\_data\_cardinality\_res () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_data\_cardinality\_res () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_data\_intersection () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_data\_intersection () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_data\_parentheses () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_data\_parentheses () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_data\_primary () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_data\_primary () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_data\_property\_iri () (owlapy.parser.DLSyntaxParser method), 140  
 visit\_data\_property\_iri () (owlapy.parser.ManchesterOWLSyntaxParser method), 138  
 visit\_data\_some\_only\_res () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_data\_some\_only\_res () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_data\_union () (owlapy.parser.DLSyntaxParser method), 139  
 visit\_data\_union () (owlapy.parser.ManchesterOWLSyntaxParser method), 137  
 visit\_data\_value\_res () (owlapy.parser.DLSyntaxParser method), 139

`visit_data_value_res()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_datatype()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_datatype()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_datatype_iri()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_datatype_iri()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_datatype_restriction()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_datatype_restriction()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_date_literal()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_date_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_datetime_literal()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_datetime_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_decimal_literal()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_decimal_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_duration_literal()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_duration_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_facet()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_facet()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_facet_restriction()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_facet_restriction()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_facet_restrictions()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_facet_restrictions()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_float_literal()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_float_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_full_iri()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_full_iri()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_has_self()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_has_self()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_individual_iri()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_individual_iri()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_individual_list()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_individual_list()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_integer_literal()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_integer_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_intersection()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_intersection()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_iri()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_iri()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_literal()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_literal_list()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_literal_list()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_non_negative_integer()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_non_negative_integer()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_object_property()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_object_property()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_object_property_iri()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_object_property_iri()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_parentheses()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_parentheses()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_primary()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_primary()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_quoted_string()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_quoted_string()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_simple_iri()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_simple_iri()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 138  
`visit_some_only_res()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_some_only_res()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_string_literal_language()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_string_literal_language()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_string_literal_no_language()` (*owlapy.parser.DLSyntaxParser method*), 140  
`visit_string_literal_no_language()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_typed_literal()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_typed_literal()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_union()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_union()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137  
`visit_value_res()` (*owlapy.parser.DLSyntaxParser method*), 139  
`visit_value_res()` (*owlapy.parser.ManchesterOWLSyntaxParser method*), 137

## W

`worst()` (*owlapy.utils.EvaluatedDescriptionSet* method), 150

## X

XSD (*in module owlapy.namespaces*), 74

XSDVocabulary (*class in owlapy.vocab*), 153