rdflib

Release 6.3.0

unknown

Mar 16, 2023

CONTENTS

RDFLib is a pure Python package for working with RDF. It contains:

- **Parsers & Serializers**
  - for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, JSON-LD, HexTuples, RDFa and Microdata
- **Store implementations**
  - memory stores
  - persistent, on-disk stores, using databases such as BerkeleyDB
  - remote SPARQL endpoints
- **Graph interface**
  - to a single graph
  - or to multiple Named Graphs within a dataset
- **SPARQL 1.1 implementation**
  - both Queries and Updates are supported

Caution: RDFLib is designed to access arbitrary network and file resources, in some cases these are directly requested resources, in other cases they are indirectly referenced resources.

If you are using RDFLib to process untrusted documents or queries you should take measures to restrict file and network access.

For information on available security measures, see the RDFLib *Security Considerations* documentation.

CHAPTER

ONE

GETTING STARTED

If you have never used RDFLib, the following will help get you started:

1.1 Getting started with RDFLib

1.1.1 Installation

RDFLib is open source and is maintained in a GitHub repository. RDFLib releases, current and previous, are listed on PyPi

The best way to install RDFLib is to use `pip` (sudo as required):

```
$ pip install rdflib
```

If you want the latest code to run, clone the `main` branch of the GitHub repo and use that or you can `pip install` directly from GitHub:

```
$ pip install git+https://github.com/RDFLib/rdflib.git@main#egg=rdflib
```

1.1.2 Support

Usage support is available via questions tagged with `[rdflib]` on StackOverflow and development support, notifications and detailed discussion through the rdflib-dev group (mailing list):

> http://groups.google.com/group/rdflib-dev

If you notice an bug or want to request an enhancement, please do so via our Issue Tracker in Github:

> http://github.com/RDFLib/rdflib/issues

1.1.3 How it all works

The package uses various Python idioms that offer an appropriate way to introduce RDF to a Python programmer who hasn't worked with RDF before.

The primary interface that RDFLib exposes for working with RDF is a *`Graph`*.

RDFLib graphs are un-sorted containers; they have ordinary Python `set` operations (e.g. *`add()`* to add a triple) plus methods that search triples and return them in arbitrary order.

RDFLib graphs also redefine certain built-in Python methods in order to behave in a predictable way. They do this by emulating container types and are best thought of as a set of 3-item tuples ("triples", in RDF-speak):

```
[
    (subject0, predicate0, object0),
    (subject1, predicate1, object1),
    ...
    (subjectN, predicateN, objectN)
 ]
```

1.1.4 A tiny example

```
from rdflib import Graph

# Create a Graph
g = Graph()

# Parse in an RDF file hosted on the Internet
g.parse("http://www.w3.org/People/Berners-Lee/card")

# Loop through each triple in the graph (subj, pred, obj)
for subj, pred, obj in g:
    # Check if there is at least one triple in the Graph
    if (subj, pred, obj) not in g:
       raise Exception("It better be!")

# Print the number of "triples" in the Graph
print(f"Graph g has {len(g)} statements.")
# Prints: Graph g has 86 statements.

# Print out the entire Graph in the RDF Turtle format
print(g.serialize(format="turtle"))
```

Here a *Graph* is created and then an RDF file online, Tim Berners-Lee's social network details, is parsed into that graph. The `print()` statement uses the `len()` function to count the number of triples in the graph.

1.1.5 A more extensive example

```
from rdflib import Graph, Literal, RDF, URIRef
# rdflib knows about quite a few popular namespaces, like W3C ontologies, schema.org etc.
from rdflib.namespace import FOAF , XSD

# Create a Graph
g = Graph()

# Create an RDF URI node to use as the subject for multiple triples
donna = URIRef("http://example.org/donna")

# Add triples using store's add() method.
g.add((donna, RDF.type, FOAF.Person))
g.add((donna, FOAF.nick, Literal("donna", lang="en")))
g.add((donna, FOAF.name, Literal("Donna Fales")))
g.add((donna, FOAF.mbox, URIRef("mailto:donna@example.org")))
```

(continues on next page)

(continued from previous page)

```
# Add another person
ed = URIRef("http://example.org/edward")

# Add triples using store's add() method.
g.add((ed, RDF.type, FOAF.Person))
g.add((ed, FOAF.nick, Literal("ed", datatype=XSD.string)))
g.add((ed, FOAF.name, Literal("Edward Scissorhands")))
g.add((ed, FOAF.mbox, Literal("e.scissorhands@example.org", datatype=XSD.anyURI)))

# Iterate over triples in store and print them out.
print("--- printing raw triples ---")
for s, p, o in g:
    print((s, p, o))

# For each foaf:Person in the store, print out their mbox property's value.
print("--- printing mboxes ---")
for person in g.subjects(RDF.type, FOAF.Person):
    for mbox in g.objects(person, FOAF.mbox):
        print(mbox)

# Bind the FOAF namespace to a prefix for more readable output
g.bind("foaf", FOAF)

# print all the data in the Notation3 format
print("--- printing mboxes ---")
print(g.serialize(format='n3'))
```

1.1.6 A SPARQL query example

```
from rdflib import Graph

# Create a Graph, pare in Internet data
g = Graph().parse("http://www.w3.org/People/Berners-Lee/card")

# Query the data in g using SPARQL
# This query returns the 'name' of all ``foaf:Person`` instances
q = """
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>

    SELECT ?name
    WHERE {
        ?p rdf:type foaf:Person .

        ?p foaf:name ?name .
    }
"""

# Apply the query to the graph and iterate through results
for r in g.query(q):
```

(continues on next page)

(continued from previous page)

```
    print(r["name"])

# prints: Timothy Berners-Lee
```

1.1.7 More examples

There are many more *examples* in the `examples` folder in the source distribution.

1.2 Loading and saving RDF

1.2.1 Reading RDF files

RDF data can be represented using various syntaxes (`turtle`, `rdf/xml`, `n3`, `n-triples`, `trix`, `JSON-LD`, etc.). The simplest format is `ntriples`, which is a triple-per-line format.

Create the file `demo.nt` in the current directory with these two lines in it:

```
<http://example.com/drewp> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
↪xmlns.com/foaf/0.1/Person> .
<http://example.com/drewp> <http://example.com/says> "Hello World" .
```

On line 1 this file says "drewp is a FOAF Person:. On line 2 it says "drep says "Hello World"".

RDFLib can guess what format the file is by the file ending (".nt" is commonly used for n-triples) so you can just use *parse()* to read in the file. If the file had a non-standard RDF file ending, you could set the keyword-parameter `format` to specify either an Internet Media Type or the format name (a *list of available parsers* is available).

In an interactive python interpreter, try this:

```
from rdflib import Graph

g = Graph()
g.parse("demo.nt")

print(len(g))
# prints: 2

import pprint
for stmt in g:
    pprint.pprint(stmt)
# prints:
# (rdflib.term.URIRef('http://example.com/drewp'),
#  rdflib.term.URIRef('http://example.com/says'),
#  rdflib.term.Literal('Hello World'))
# (rdflib.term.URIRef('http://example.com/drewp'),
#  rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
#  rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Person'))
```

The final lines show how RDFLib represents the two statements in the file: the statements themselves are just length-3 tuples ("triples") and the subjects, predicates, and objects of the triples are all rdflib types.

1.2.2 Reading remote RDF

Reading graphs from the Internet is easy:

```
from rdflib import Graph

g = Graph()
g.parse("http://www.w3.org/People/Berners-Lee/card")
print(len(g))
# prints: 86
```

rdflib.Graph.parse() can process local files, remote data via a URL, as in this example, or RDF data in a string (using the `data` parameter).

1.2.3 Saving RDF

To store a graph in a file, use the *rdflib.Graph.serialize()* function:

```
from rdflib import Graph

g = Graph()
g.parse("http://www.w3.org/People/Berners-Lee/card")
g.serialize(destination="tbl.ttl")
```

This parses data from http://www.w3.org/People/Berners-Lee/card and stores it in a file `tbl.ttl` in this directory using the turtle format, which is the default RDF serialization (as of rdflib 6.0.0).

To read the same data and to save it as an RDF/XML format string in the variable `v`, do this:

```
from rdflib import Graph

g = Graph()
g.parse("http://www.w3.org/People/Berners-Lee/card")
v = g.serialize(format="xml")
```

The following table lists the RDF formats you can serialize data to with rdflib, out of the box, and the `format=KEYWORD` keyword used to reference them within `serialize()`:

RDF Format	Keyword	Notes
Turtle	turtle, ttl or turtle2	turtle2 is just turtle with more spacing & linebreaks
RDF/XML	xml or pretty-xml	Was the default format, rdflib < 6.0.0
JSON-LD	json-ld	There are further options for compact syntax and other JSON-LD variants
N-Triples	ntriples, nt or nt11	nt11 is exactly like nt, only utf8 encoded
Notation-3	n3	N3 is a superset of Turtle that also caters for rules and a few other things
Trig	trig	Turtle-like format for RDF triples + context (RDF quads) and thus multiple graphs
Trix	trix	RDF/XML-like format for RDF quads
N-Quads	nquads	N-Triples-like format for RDF quads

1.2.4 Working with multi-graphs

To read and query multi-graphs, that is RDF data that is context-aware, you need to use rdflib's *rdflib.ConjunctiveGraph* or *rdflib.Dataset* class. These are extensions to *rdflib.Graph* that know all about quads (triples + graph IDs).

If you had this multi-graph data file (in the `trig` format, using new-style `PREFIX` statement (not the older `@prefix`):

```
PREFIX eg: <http://example.com/person/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

eg:graph-1 {
    eg:drewp a foaf:Person .
    eg:drewp eg:says "Hello World" .
}

eg:graph-2 {
    eg:nick a foaf:Person .
    eg:nick eg:says "Hi World" .
}
```

You could parse the file and query it like this:

```
from rdflib import Dataset
from rdflib.namespace import RDF

g = Dataset()
g.parse("demo.trig")

for s, p, o, g in g.quads((None, RDF.type, None, None)):
    print(s, g)
```

This will print out:

```
http://example.com/person/drewp http://example.com/person/graph-1
http://example.com/person/nick http://example.com/person/graph-2
```

1.3 Creating RDF triples

1.3.1 Creating Nodes

RDF data is a graph where the nodes are URI references, Blank Nodes or Literals. In RDFLib, these node types are represented by the classes *URIRef*, *BNode*, and *Literal*. `URIRefs` and `BNodes` can both be thought of as resources, such a person, a company, a website, etc.

- A `BNode` is a node where the exact URI is not known - usually a node with identity only in relation to other nodes.
- A `URIRef` is a node where the exact URI is known. In addition to representing some subjects and predicates in RDF graphs, `URIRefs` are always used to represent properties/predicates
- `Literals` represent object values, such as a name, a date, a number, etc. The most common literal values are XML data types, e.g. string, int… but custom types can be declared too

Nodes can be created by the constructors of the node classes:

```
from rdflib import URIRef, BNode, Literal

bob = URIRef("http://example.org/people/Bob")
linda = BNode()  # a GUID is generated

name = Literal("Bob")  # passing a string
age = Literal(24)  # passing a python int
height = Literal(76.5)  # passing a python float
```

Literals can be created from Python objects, this creates `data-typed literals`. For the details on the mapping see *Literals*.

For creating many `URIRefs` in the same `namespace`, i.e. URIs with the same prefix, RDFLib has the *rdflib.namespace.Namespace* class

```
from rdflib import Namespace

n = Namespace("http://example.org/people/")

n.bob  # == rdflib.term.URIRef("http://example.org/people/bob")
n.eve  # == rdflib.term.URIRef("http://example.org/people/eve")
```

This is very useful for schemas where all properties and classes have the same URI prefix. RDFLib defines Namespaces for some common RDF/OWL schemas, including most W3C ones:

```
from rdflib.namespace import CSVW, DC, DCAT, DCTERMS, DOAP, FOAF, ODRL2, ORG, OWL, \
                             PROF, PROV, RDF, RDFS, SDO, SH, SKOS, SOSA, SSN, TIME, \
                             VOID, XMLNS, XSD

RDF.type
# == rdflib.term.URIRef("http://www.w3.org/1999/02/22-rdf-syntax-ns#type")

FOAF.knows
# == rdflib.term.URIRef("http://xmlns.com/foaf/0.1/knows")

PROF.isProfileOf
# == rdflib.term.URIRef("http://www.w3.org/ns/dx/prof/isProfileOf")

SOSA.Sensor
# == rdflib.term.URIRef("http://www.w3.org/ns/sosa/Sensor")
```

1.3.2 Adding Triples to a graph

We already saw in *Loading and saving RDF*, how triples can be added from files and online locations with with the *parse()* function.

Triples can also be added within Python code directly, using the *add()* function:

`Graph.`**`add`**`(`*triple*`)`

Add a triple with self as context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –

- **triple** (Tuple[*Node*, *Node*, *Node*]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

add() takes a 3-tuple (a "triple") of RDFLib nodes. Using the nodes and namespaces we defined previously:

```
from rdflib import Graph, URIRef, Literal, BNode
from rdflib.namespace import FOAF, RDF

g = Graph()
g.bind("foaf", FOAF)

bob = URIRef("http://example.org/people/Bob")
linda = BNode()  # a GUID is generated

name = Literal("Bob")
age = Literal(24)

g.add((bob, RDF.type, FOAF.Person))
g.add((bob, FOAF.name, name))
g.add((bob, FOAF.age, age))
g.add((bob, FOAF.knows, linda))
g.add((linda, RDF.type, FOAF.Person))
g.add((linda, FOAF.name, Literal("Linda")))

print(g.serialize())
```

outputs:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/people/Bob> a foaf:Person ;
    foaf:age 24 ;
    foaf:knows [ a foaf:Person ;
            foaf:name "Linda" ] ;
    foaf:name "Bob" .
```

For some properties, only one value per resource makes sense (i.e they are *functional properties*, or have a max-cardinality of 1). The *set()* method is useful for this:

```
from rdflib import Graph, URIRef, Literal
from rdflib.namespace import FOAF

g = Graph()
bob = URIRef("http://example.org/people/Bob")

g.add((bob, FOAF.age, Literal(42)))
print(f"Bob is {g.value(bob, FOAF.age)}")
# prints: Bob is 42

g.set((bob, FOAF.age, Literal(43)))  # replaces 42 set above
print(f"Bob is now {g.value(bob, FOAF.age)}")
# prints: Bob is now 43
```

rdflib.graph.Graph.value() is the matching query method. It will return a single value for a property, optionally raising an exception if there are more.

You can also add triples by combining entire graphs, see *Set Operations on RDFLib Graphs*.

1.3.3 Removing Triples

Similarly, triples can be removed by a call to *remove()*:

Graph.**remove**(*triple*)

Remove a triple from the graph

If the triple does not provide a context attribute, removes the triple from all contexts.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type

TypeVar(_GraphT, bound= *Graph*)

When removing, it is possible to leave parts of the triple unspecified (i.e. passing None), this will remove all matching triples:

```
g.remove((bob, None, None))  # remove all triples about bob
```

1.3.4 An example

LiveJournal produces FOAF data for their users, but they seem to use foaf:member_name for a person's full name but foaf:member_name isn't in FOAF's namespace and perhaps they should have used foaf:name

To retrieve some LiveJournal data, add a foaf:name for every foaf:member_name and then remove the foaf:member_name values to ensure the data actually aligns with other FOAF data, we could do this:

```
from rdflib import Graph
from rdflib.namespace import FOAF

g = Graph()
# get the data
g.parse("http://danbri.livejournal.com/data/foaf")

# for every foaf:member_name, add foaf:name and remove foaf:member_name
for s, p, o in g.triples((None, FOAF['member_name'], None)):
    g.add((s, FOAF['name'], o))
    g.remove((s, FOAF['member_name'], o))
```

Note: Since rdflib 5.0.0, using foaf:member_name is somewhat prevented in RDFlib since FOAF is declared as a *ClosedNamespace()* class instance that has a closed set of members and foaf:member_name isn't one of them! If LiveJournal had used RDFlib 5.0.0, an error would have been raised for foaf:member_name when the triple was created.

1.3.5 Creating Containers & Collections

There are two convenience classes for RDF Containers & Collections which you can use instead of declaring each triple of a Containers or a Collections individually:

- *Container()* (also `Bag`, `Seq` & `Alt`) and
- *Collection()*

See their documentation for how.

1.4 Navigating Graphs

An RDF Graph is a set of RDF triples, and we try to mirror exactly this in RDFLib. The Python *Graph()* tries to emulate a container type.

1.4.1 Graphs as Iterators

RDFLib graphs override *__iter__()* in order to support iteration over the contained triples:

```
for s, p, o in someGraph:
    if not (s, p, o) in someGraph:
        raise Exception("Iterator / Container Protocols are Broken!!")
```

This loop iterates through all the subjects(s), predicates (p) & objects (o) in `someGraph`.

1.4.2 Contains check

Graphs implement *__contains__()*, so you can check if a triple is in a graph with a `triple in graph` syntax:

```
from rdflib import URIRef
from rdflib.namespace import RDF

bob = URIRef("http://example.org/people/bob")
if (bob, RDF.type, FOAF.Person) in graph:
    print("This graph knows that Bob is a person!")
```

Note that this triple does not have to be completely bound:

```
if (bob, None, None) in graph:
    print("This graph contains triples about Bob!")
```

1.4.3 Set Operations on RDFLib Graphs

Graphs override several pythons operators: *__iadd__()*, *__isub__()*, etc. This supports addition, subtraction and other set-operations on Graphs:

operation	effect
G1 + G2	return new graph with union (triples on both)
G1 += G2	in place union / addition
G1 - G2	return new graph with difference (triples in G1, not in G2)
G1 -= G2	in place difference / subtraction
G1 & G2	intersection (triples in both graphs)
G1 ^ G2	xor (triples in either G1 or G2, but not in both)

Warning: Set-operations on graphs assume Blank Nodes are shared between graphs. This may or may not be what you want. See *Merging graphs* for details.

1.4.4 Basic Triple Matching

Instead of iterating through all triples, RDFLib graphs support basic triple pattern matching with a *triples()* function. This function is a generator of triples that match a pattern given by arguments, i.e. arguments restrict the triples that are returned. Terms that are None are treated as a wildcard. For example:

```
g.parse("some_foaf.ttl")
# find all subjects (s) of type (rdf:type) person (foaf:Person)
for s, p, o in g.triples((None, RDF.type, FOAF.Person)):
    print(f"{s} is a person")

# find all subjects of any type
for s, p, o in g.triples((None,  RDF.type, None)):
    print(f"{s} is a {o}")

# create a graph
bobgraph = Graph()
# add all triples with subject 'bob'
bobgraph += g.triples((bob, None, None))
```

If you are not interested in whole triples, you can get only the bits you want with the methods *objects()*, *subjects()*, *predicates()*, *predicate_objects()*, etc. Each take parameters for the components of the triple to constraint:

```
for person in g.subjects(RDF.type, FOAF.Person):
    print("{} is a person".format(person))
```

Finally, for some properties, only one value per resource makes sense (i.e they are *functional properties*, or have a max-cardinality of 1). The *value()* method is useful for this, as it returns just a single node, not a generator:

```
# get any name of bob
name = g.value(bob, FOAF.name)
# get the one person that knows bob and raise an exception if more are found
mbox = g.value(predicate = FOAF.name, object=bob, any=False)
```

1.4.5 `Graph` methods for accessing triples

Here is a list of all convenience methods for querying Graphs:

`Graph.triples`(*triple: Tuple[Optional[Node], Optional[Node], Optional[Node]]*) → Generator[Tuple[*Node*, *Node*, *Node*], None, None]

`Graph.triples`(*triple: Tuple[Optional[Node], Path, Optional[Node]]*) → Generator[Tuple[*Node*, *Path*, *Node*], None, None]

`Graph.triples`(*triple: Tuple[Optional[Node], Optional[Union[Path, Node]], Optional[Node]]*) → Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

Generator over the triple store

Returns triples that match the given triple pattern. If triple pattern does not provide a context, all contexts will be searched.

Parameters
triple (Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]]) –

Return type
Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

`Graph.value`(*subject: None = None*, *predicate: None = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: Optional[Node] = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

`Graph.value`(*subject: Optional[Node] = None*, *predicate: None = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: None = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

`Graph.value`(*subject: None = None*, *predicate: Optional[Node] = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: None = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

`Graph.value`(*subject: Optional[Node] = None*, *predicate: Optional[Node] = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: Optional[Node] = None*, *default: Optional[Node] = None*, *any: bool = True*) → Optional[*Node*]

Get a value for a pair of two criteria

Exactly one of subject, predicate, object must be None. Useful if one knows that there may only be one value.

It is one of those situations that occur a lot, hence this 'macro' like utility

Parameters: subject, predicate, object – exactly one must be None default – value to be returned if no values found any – if True, return any value in the case there is more than one, else, raise UniquenessError

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –
- **default** (Optional[*Node*]) –
- **any** (bool) –

Return type
Optional[*Node*]

`Graph.subjects`(*predicate=None*, *object=None*, *unique=False*)

A generator of (optionally unique) subjects with the given predicate and object

Parameters

- **predicate** (Union[None, *Path*, *Node*]) –
- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

Graph.**objects**(*subject=None*, *predicate=None*, *unique=False*)

A generator of (optionally unique) objects with the given subject and predicate

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Union[None, *Path*, *Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

Graph.**predicates**(*subject=None*, *object=None*, *unique=False*)

A generator of (optionally unique) predicates with the given subject and object

Parameters

- **subject** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

Graph.**subject_objects**(*predicate=None*, *unique=False*)

A generator of (optionally unique) (subject, object) tuples for the given predicate

Parameters

- **predicate** (Union[None, *Path*, *Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

Graph.**subject_predicates**(*object=None*, *unique=False*)

A generator of (optionally unique) (subject, predicate) tuples for the given object

Parameters

- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

Graph.**predicate_objects**(*subject=None*, *unique=False*)

A generator of (optionally unique) (predicate, object) tuples for the given subject

Parameters

- **subject** (Optional[*Node*]) –
- **unique** (bool) –

Return type

Generator[Tuple[*Node*, *Node*], None, None]

1.5 Querying with SPARQL

1.5.1 Run a Query

The RDFLib comes with an implementation of the SPARQL 1.1 Query and SPARQL 1.1 Update query languages.

Queries can be evaluated against a graph with the *rdflib.graph.Graph.query()* method, and updates with *rdflib.graph.Graph.update()*.

The query method returns a *rdflib.query.Result* instance. For SELECT queries, iterating over this returns *rdflib.query.ResultRow* instances, each containing a set of variable bindings. For `CONSTRUCT`/`DESCRIBE` queries, iterating over the result object gives the triples. For `ASK` queries, iterating will yield the single boolean answer, or evaluating the result object in a boolean-context (i.e. `bool(result)`)

For example…

```
import rdflib
g = rdflib.Graph()
g.parse("http://danbri.org/foaf.rdf#")

knows_query = """
SELECT DISTINCT ?aname ?bname
WHERE {
    ?a foaf:knows ?b .
    ?a foaf:name ?aname .
    ?b foaf:name ?bname .
}"""

qres = g.query(knows_query)
for row in qres:
    print(f"{row.aname} knows {row.bname}")
```

The results are tuples of values in the same order as your `SELECT` arguments. Alternatively, the values can be accessed by variable name, either as attributes, or as items, e.g. `row.b` and `row["b"]` are equivalent. The above, given the appropriate data, would print something like:

```
Timothy Berners-Lee knows Edd Dumbill
Timothy Berners-Lee knows Jennifer Golbeck
Timothy Berners-Lee knows Nicholas Gibbins
...
```

As an alternative to using `SPARQL`s `PREFIX`, namespace bindings can be passed in with the `initNs` kwarg, see *Namespaces and Bindings*.

Variables can also be pre-bound, using the `initBindings` kwarg which can pass in a `dict` of initial bindings. This is particularly useful for prepared queries, as described below.

1.5.2 Update Queries

Update queries are performed just like reading queries but using the *rdflib.graph.Graph.update()* method. An example:

```
from rdflib import Graph

# Create a Graph, add in some test data
g = Graph()
g.parse(
    data="""
        <x:> a <c:> .
        <y:> a <c:> .
    """,
    format="turtle"
)

# Select all the things (s) that are of type (rdf:type) c:
qres = g.query("""SELECT ?s WHERE { ?s a <c:> }""")

for row in qres:
    print(f"{row.s}")
# prints:
# x:
# y:

# Add in a new triple using SPATQL UPDATE
g.update("""INSERT DATA { <z:> a <c:> }""")

# Select all the things (s) that are of type (rdf:type) c:
qres = g.query("""SELECT ?s WHERE { ?s a <c:> }""")

print("After update:")
for row in qres:
    print(f"{row.s}")
# prints:
# x:
# y:
# z:

# Change type of <y:> from <c:> to <d:>
g.update("""
         DELETE { <y:> a <c:> }
         INSERT { <y:> a <d:> }
         WHERE { <y:> a <c:> }
         """)
print("After second update:")
qres = g.query("""SELECT ?s ?o WHERE { ?s a ?o }""")
for row in qres:
    print(f"{row.s} a {row.o}")
```

(continues on next page)

(continued from previous page)

```
# prints:
# x: a c:
# z: a c:
# y: a d:
```

1.5.3 Querying a Remote Service

The `SERVICE` keyword of SPARQL 1.1 can send a query to a remote SPARQL endpoint.

```
import rdflib

g = rdflib.Graph()
qres = g.query(
    """
    SELECT ?s
    WHERE {
      SERVICE <http://dbpedia.org/sparql> {
        ?s a ?o .
      }
    }
    LIMIT 3
    """
)

for row in qres:
    print(row.s)
```

This example sends a query to DBPedia's SPARQL endpoint service so that it can run the query and then send back the result:

```
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.openlinksw.com/schemas/
↪virtcxml#FacetCategoryPattern>
<http://www.w3.org/2001/XMLSchema#anyURI> <http://www.w3.org/2000/01/rdf-schema#Datatype>
<http://www.w3.org/2001/XMLSchema#anyURI> <http://www.w3.org/2000/01/rdf-schema#Datatype>
```

1.5.4 Prepared Queries

RDFLib lets you *prepare* queries before execution, this saves re-parsing and translating the query into SPARQL Algebra each time.

The method *rdflib.plugins.sparql.prepareQuery()* takes a query as a string and will return a *rdflib.plugins.sparql.sparql.Query* object. This can then be passed to the *rdflib.graph.Graph.query()* method.

The `initBindings` kwarg can be used to pass in a `dict` of initial bindings:

```
q = prepareQuery(
    "SELECT ?s WHERE { ?person foaf:knows ?s .}",
    initNs = { "foaf": FOAF }
)

g = rdflib.Graph()
```

(continues on next page)

(continued from previous page)

```
g.parse("foaf.rdf")

tim = rdflib.URIRef("http://www.w3.org/People/Berners-Lee/card#i")

for row in g.query(q, initBindings={'person': tim}):
    print(row)
```

1.5.5 Custom Evaluation Functions

For experts, it is possible to override how bits of SPARQL algebra are evaluated. By using the setuptools entry-point `rdf.plugins.sparqleval`, or simply adding to an entry to *`rdflib.plugins.sparql.CUSTOM_EVALS`*, a custom function can be registered. The function will be called for each algebra component and may raise `NotImplementedError` to indicate that this part should be handled by the default implementation.

See `examples/custom_eval.py`

1.6 Utilities & convenience functions

For RDF programming, RDFLib and Python may not be the fastest tools, but we try hard to make them the easiest and most convenient to use and thus the *fastest* overall!

This is a collection of hints and pointers for hassle-free RDF coding.

1.6.1 Functional properties

Use *`value()`* and *`set()`* to work with *functional property* instances, i.e. properties than can only occur once for a resource.

```
from rdflib import Graph, URIRef, Literal, BNode
from rdflib.namespace import FOAF, RDF

g = Graph()
g.bind("foaf", FOAF)

# Add demo data
bob = URIRef("http://example.org/people/Bob")
g.add((bob, RDF.type, FOAF.Person))
g.add((bob, FOAF.name, Literal("Bob")))
g.add((bob, FOAF.age, Literal(38)))

# To get a single value, use 'value'
print(g.value(bob, FOAF.age))
# prints: 38

# To change a single of value, use 'set'
g.set((bob, FOAF.age, Literal(39)))
print(g.value(bob, FOAF.age))
# prints: 39
```

1.6.2 Slicing graphs

Python allows slicing arrays with a `slice` object, a triple of `start`, `stop` and `step-size`:

```
for i in range(20)[2:9:3]:
    print(i)
# prints:
# 2, 5, 8
```

RDFLib graphs override `__getitem__` and we pervert the slice triple to be a RDF triple instead. This lets slice syntax be a shortcut for *triples()*, *subject_predicates()*, *__contains__()*, and other Graph query-methods:

```
from rdflib import Graph, URIRef, Literal, BNode
from rdflib.namespace import FOAF, RDF

g = Graph()
g.bind("foaf", FOAF)

# Add demo data
bob = URIRef("http://example.org/people/Bob")
bill = URIRef("http://example.org/people/Bill")
g.add((bob, RDF.type, FOAF.Person))
g.add((bob, FOAF.name, Literal("Bob")))
g.add((bob, FOAF.age, Literal(38)))
g.add((bob, FOAF.knows, bill))

print(g[:])
# same as
print(iter(g))

print(g[bob])
# same as
print(g.predicate_objects(bob))

print(g[bob: FOAF.knows])
# same as
print(g.objects(bob, FOAF.knows))

print(g[bob: FOAF.knows: bill])
# same as
print((bob, FOAF.knows, bill) in g)

print(g[:FOAF.knows])
# same as
print(g.subject_objects(FOAF.knows))
```

See *examples.slice* for a complete example.

Note: Slicing is convenient for run-once scripts for playing around in the Python REPL, however since slicing returns tuples of varying length depending on which parts of the slice are bound, you should be careful using it in more complicated programs. If you pass in variables, and they are `None` or `False`, you may suddenly get a generator of different length tuples back than you expect.

1.6.3 SPARQL Paths

SPARQL property paths are possible using overridden operators on URIRefs. See *examples.foafpaths* and *rdflib.paths*.

1.6.4 Serializing a single term to N3

For simple output, or simple serialisation, you often want a nice readable representation of a term. All terms (URIRef, Literal etc.) have a `n3`, method, which will return a suitable N3 format:

```
from rdflib import Graph, URIRef, Literal
from rdflib.namespace import FOAF

# A URIRef
person = URIRef("http://xmlns.com/foaf/0.1/Person")
print(person.n3())
# prints: <http://xmlns.com/foaf/0.1/Person>

# Simplifying the output with a namespace prefix:
g = Graph()
g.bind("foaf", FOAF)

print(person.n3(g.namespace_manager))
# prints foaf:Person

# A typed literal
l = Literal(2)
print(l.n3())
# prints "2"^^<http://www.w3.org/2001/XMLSchema#integer>

# Simplifying the output with a namespace prefix
# XSD is built in, so no need to bind() it!
l.n3(g.namespace_manager)
# prints: "2"^^xsd:integer
```

1.6.5 Parsing data from a string

You can parse data from a string with the `data` param:

```
from rdflib import Graph

g = Graph().parse(data="<a:> <p:> <p:>.")
for r in g.triples((None, None, None)):
    print(r)
# prints: (rdflib.term.URIRef('a:'), rdflib.term.URIRef('p:'), rdflib.term.URIRef('p:'))
```

1.6.6 Command Line tools

RDFLib includes a handful of commandline tools, see *rdflib.tools*.

1.7 examples Package

These examples all live in `./examples` in the source-distribution of RDFLib.

1.7.1 conjunctive_graphs Module

An RDFLib ConjunctiveGraph is an (unnamed) aggregation of all the Named Graphs within a Store. The *get_context()* method can be used to get a particular named graph for use, such as to add triples to, or the default graph can be used.

This example shows how to create Named Graphs and work with the conjunction (union) of all the graphs.

1.7.2 custom_datatype Module

RDFLib can map between RDF data-typed literals and Python objects.

Mapping for integers, floats, dateTimes, etc. are already added, but you can also add your own.

This example shows how *rdflib.term.bind()* lets you register new mappings between literal datatypes and Python objects

1.7.3 custom_eval Module

This example shows how a custom evaluation function can be added to handle certain SPARQL Algebra elements.

A custom function is added that adds `rdfs:subClassOf` "inference" when asking for `rdf:type` triples.

Here the custom eval function is added manually, normally you would use setuptools and entry_points to do it: i.e. in your setup.py:

```
entry_points = {
    'rdf.plugins.sparqleval': [
        'myfunc = mypackage:MyFunction',
        ],
}
```

examples.custom_eval.**customEval**(*ctx*, *part*)

Rewrite triple patterns to get super-classes

1.7.4 foafpaths Module

SPARQL 1.1 defines path operators for combining/repeating predicates in triple-patterns.

We overload some Python operators on URIRefs to allow creating path operators directly in Python.

Operator	Path
`p1 / p2`	Path sequence
`p1 | p2`	Path alternative
`p1 * '*'`	chain of 0 or more p's
`p1 * '+'`	chain of 1 or more p's
`p1 * '?'`	0 or 1 p
`~p1`	p1 inverted, i.e. (s p1 o) <=> (o ~p1 s)
`-p1`	NOT p1, i.e. any property but p1

These can then be used in property position for `s,p,o` triple queries for any graph method.

See the docs for *rdflib.paths* for the details.

This example shows how to get the name of friends (i.e values two steps away x knows y, y name z) with a single query.

1.7.5 prepared_query Module

SPARQL Queries be prepared (i.e parsed and translated to SPARQL algebra) by the *rdflib.plugins.sparql.prepareQuery()* method.

`initNs` can be used instead of PREFIX values.

When executing, variables can be bound with the `initBindings` keyword parameter.

1.7.6 resource_example Module

RDFLib has a *Resource* class, for a resource-centric API. The *Graph* class also has a `resource` function that can be used to create resources and manipulate them by quickly adding or querying for triples where this resource is the subject.

This example shows g.resource() in action.

1.7.7 berkeleydb_example Module

BerkeleyDB in use as a persistent Graph store.

Example 1: simple actions

- creating a ConjunctiveGraph using the BerkeleyDB Store
- adding triples to it
- counting them
- closing the store, emptying the graph
- re-opening the store using the same DB files
- getting the same count of triples as before

Example 2: larger data

- loads multiple graphs downloaded from GitHub into a BerkeleyDB-baked graph stored in the folder gsq_vocabs.
- does not delete the DB at the end so you can see it on disk

`examples.berkeleydb_example.example_1()`

Creates a ConjunctiveGraph and performs some BerkeleyDB tasks with it

`examples.berkeleydb_example.example_2()`

Loads a number of SKOS vocabularies from GitHub into a BerkeleyDB-backed graph stored in the local folder 'gsq_vocabs'

Should print out the number of triples after each load, e.g.:

177 248 289 379 421 628 764 813 965 1381 9666 9719 …

1.7.8 `slice` Module

RDFLib Graphs (and Resources) can be "sliced" with [] syntax

This is a short-hand for iterating over triples.

Combined with SPARQL paths (see `foafpaths.py`) - quite complex queries can be realised.

See *rdflib.graph.Graph.__getitem__()* for details

1.7.9 `smushing` Module

A FOAF smushing example.

Filter a graph by normalizing all `foaf:Persons` into URIs based on their `mbox_sha1sum`.

Suppose I get two FOAF documents each talking about the same person (according to `mbox_sha1sum`) but they each used a *rdflib.term.BNode* for the subject. For this demo I've combined those two documents into one file:

This filters a graph by changing every subject with a `foaf:mbox_sha1sum` into a new subject whose URI is based on the `sha1sum`. This new graph might be easier to do some operations on.

An advantage of this approach over other methods for collapsing BNodes is that I can incrementally process new FOAF documents as they come in without having to access my ever-growing archive. Even if another `65b983bb397fb71849da910996741752ace8369b` document comes in next year, I would still give it the same stable subject URI that merges with my existing data.

1.7.10 `sparql_query_example` Module

SPARQL Query using *rdflib.graph.Graph.query()*

The method returns a *Result*, iterating over this yields *ResultRow* objects

The variable bindings can be accessed as attributes of the row objects For variable names that are not valid python identifiers, dict access (i.e. with `row[var] / __getitem__`) is also possible.

vars contains the variables

1.7.11 sparql_update_example Module

SPARQL Update statements can be applied with *rdflib.graph.Graph.update()*

1.7.12 sparqlstore_example Module

Simple examples showing how to use the SPARQLStore

1.7.13 swap_primer Module

This is a simple primer using some of the example stuff in the Primer on N3:

http://www.w3.org/2000/10/swap/Primer

1.7.14 transitive Module

An example illustrating how to use the *transitive_subjects()* and *transitive_objects()* graph methods

Formal definition

The *transitive_objects()* method finds all nodes such that there is a path from subject to one of those nodes using only the predicate property in the triples. The *transitive_subjects()* method is similar; it finds all nodes such that there is a path from the node to the object using only the predicate property.

Informal description, with an example

In brief, *transitive_objects()* walks forward in a graph using a particular property, and *transitive_subjects()* walks backward. A good example uses a property `ex:parent`, the semantics of which are biological parentage. The *transitive_objects()* method would get all the ancestors of a particular person (all nodes such that there is a parent path between the person and the object). The *transitive_subjects()* method would get all the descendants of a particular person (all nodes such that there is a parent path between the node and the person). So, say that your URI is `ex:person`.

This example would get all of your (known) ancestors, and then get all the (known) descendants of your maternal grandmother.

> **Warning:** The *transitive_objects()* method has the start node as the *first* argument, but the *transitive_subjects()* method has the start node as the *second* argument.

User-defined transitive closures

The method *transitiveClosure()* returns transtive closures of user-defined functions.

1.7.15 secure_with_audit Module

This example demonstrates how to use Python audit hooks to block access to files and URLs.

It installs a audit hook with sys.addaudithook that blocks access to files and URLs that end with `blocked.jsonld`.

The code in the example then verifies that the audit hook is blocking access to URLs and files as expected.

examples.secure_with_audit.**audit_hook**(*name*, *args*)

An audit hook that blocks access when an attempt is made to open a file or URL that ends with `blocked.jsonld`.

Details of the audit events can be seen in the audit events table.

Parameters

- **name** (str) – The name of the audit event.
- **args** (Tuple[Any, ...]) – The arguments of the audit event.

Return type
None

Returns
None if the audit hook does not block access.

Raises
PermissionError – If the file or URL being accessed ends with `blocked.jsonld`.

examples.secure_with_audit.**main**()

The main code of the example.

The important steps are:

- Install a custom audit hook that blocks some URLs and files.
- Attempt to parse a JSON-LD document that will result in a blocked URL being accessed.
- Verify that the audit hook blocked access to the URL.
- Attempt to parse a JSON-LD document that will result in a blocked file being accessed.
- Verify that the audit hook blocked access to the file.

Return type
None

1.7.16 secure_with_urlopen Module

This example demonstrates how to use a custom global URL opener installed with `urllib.request.install_opener` to block access to URLs.

class examples.secure_with_urlopen.**SecuredHTTPHandler**(*debuglevel=0*)

Bases: HTTPHandler

A HTTP handler that blocks access to URLs that end with "blocked.jsonld".

__module__ = 'examples.secure_with_urlopen'

http_open(*req*)

Block access to URLs that end with "blocked.jsonld".

Parameters
req (Request) – The request to open.

Return type
HTTPResponse

Returns
The response.

Raises
PermissionError – If the URL ends with "blocked.jsonld".

examples.secure_with_urlopen.**main**()

The main code of the example.

The important steps are:

- Install a custom global URL opener that blocks some URLs.
- Attempt to parse a JSON-LD document that will result in a blocked URL being accessed.
- Verify that the URL opener blocked access to the URL.

Return type
None

CHAPTER

TWO

IN DEPTH

If you are familiar with RDF and are looking for details on how RDFLib handles it, these are for you:

2.1 RDF terms in rdflib

Terms are the kinds of objects that can appear in a RDFLib's graph's triples. Those that are part of core RDF concepts are: `IRIs`, `Blank Node` and `Literal`, the latter consisting of a literal value and either a datatype or an **RFC 3066** language tag.

Note: RDFLib's class for representing IRIs/URIs is called "URIRef" because, at the time it was implemented, that was what the then current RDF specification called URIs/IRIs. We preserve that class name but refer to the RDF object as "IRI".

2.1.1 Class hierarchy

All terms in RDFLib are sub-classes of the *rdflib.term.Identifier* class. A class diagram of the various terms is:

Fig. 1: Term Class Hierarchy

Nodes are a subset of the Terms that underlying stores actually persist.

The set of such Terms depends on whether or not the store is formula-aware. Stores that aren't formula-aware only persist those terms core to the RDF Model but those that are formula-aware also persist the N3 extensions. However, utility terms that only serve the purpose of matching nodes by term-patterns will probably only be terms and not nodes.

2.1.2 Python Classes

The three main RDF objects - *IRI*, *Blank Node* and *Literal* are represented in RDFLib by these three main Python classes:

URIRef

An IRI (Internationalized Resource Identifier) is represented within RDFLib using the URIRef class. From the RDF 1.1 specification's IRI section:

Here is the *URIRef* class' auto-built documentation:

class rdflib.term.**URIRef**(*value: str*, *base: Optional[str] = None*)

RDF 1.1's IRI Section https://www.w3.org/TR/rdf11-concepts/#section-IRIs

---

Note: Documentation on RDF outside of RDFLib uses the term IRI or URI whereas this class is called URIRef. This is because it was made when the first version of the RDF specification was current, and it used the term *URIRef*, see RDF 1.0 URIRef

---

An IRI (Internationalized Resource Identifier) within an RDF graph is a Unicode string that conforms to the syntax defined in RFC 3987.

IRIs in the RDF abstract syntax MUST be absolute, and MAY contain a fragment identifier.

IRIs are a generalization of URIs [RFC3986] that permits a wider range of Unicode characters.

```
>>> from rdflib import URIRef
>>> uri = URIRef()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __new__() missing 1 required positional argument: 'value'
>>> uri = URIRef('')
>>> uri
rdflib.term.URIRef('')
>>> uri = URIRef('http://example.com')
>>> uri
rdflib.term.URIRef('http://example.com')
>>> uri.n3()
'<http://example.com>'
```

BNodes

In RDF, a blank node (also called BNode) is a node in an RDF graph representing a resource for which an IRI or literal is not given. The resource represented by a blank node is also called an anonymous resource. According to the RDF standard, a blank node can only be used as subject or object in a triple, although in some syntaxes like Notation 3 it is acceptable to use a blank node as a predicate. If a blank node has a node ID (not all blank nodes are labelled in all RDF serializations), it is limited in scope to a particular serialization of the RDF graph, i.e. the node p1 in one graph does not represent the same node as a node named p1 in any other graph – wikipedia

Here is the *BNode* class' auto-built documentation:

class rdflib.term.**BNode**(*value: ~typing.Optional[str] = None*, *_sn_gen: ~typing.Callable[[], str] = <function _serial_number_generator.<locals>._generator>*, *_prefix: str = 'N'*)

RDF 1.1's Blank Nodes Section: https://www.w3.org/TR/rdf11-concepts/#section-blank-nodes

Blank Nodes are local identifiers for unnamed nodes in RDF graphs that are used in some concrete RDF syntaxes or RDF store implementations. They are always locally scoped to the file or RDF store, and are not persistent or portable identifiers for blank nodes. The identifiers for Blank Nodes are not part of the RDF abstract syntax, but are entirely dependent on particular concrete syntax or implementation (such as Turtle, JSON-LD).

—

RDFLib's `BNode` class makes unique IDs for all the Blank Nodes in a Graph but you should *never* expect, or reply on, BNodes' IDs to match across graphs, or even for multiple copies of the same graph, if they are regenerated from some non-RDFLib source, such as loading from RDF data.

```
>>> from rdflib import BNode
>>> bn = BNode()
>>> bn
rdflib.term.BNode('AFwALAKU0')
>>> bn.n3()
'_:AFwALAKU0'
```

Literals

Literals are attribute values in RDF, for instance, a person's name, the date of birth, height, etc. and are stored using simple data types, e.g. *string*, *double*, *dateTime* etc. This usually looks something like this:

```
name = Literal("Nicholas")  # the name 'Nicholas', as a string

age = Literal(39, datatype=XSD.integer)  # the number 39, as an integer
```

A slightly special case is a *langString* which is a *string* with a language tag, e.g.:

```
name = Literal("Nicholas", lang="en")  # the name 'Nicholas', as an English string
imie = Literal("Mikołaj", lang="pl")  # the Polish version of the name 'Nicholas'
```

Special literal types indicated by use of a custom IRI for a literal's `datatype` value, for example the GeoSPARQL RDF standard invents a custom datatype, `geoJSONLiteral` to indicate GeoJSON geometry serlializations like this:

```
GEO = Namespace("http://www.opengis.net/ont/geosparql#")

geojson_geometry = Literal(
    '{"type": "Point", "coordinates": [-83.38,33.95]}',
    datatype=GEO.geoJSONLiteral
```

Here is the `Literal` class' auto-built documentation, followed by notes on Literal from the RDF 1.1 specification 'Literals' section.

class rdflib.term.**Literal**(*lexical_or_value: Any*, *lang: Optional[str] = None*, *datatype: Optional[str] = None*, *normalize: Optional[bool] = None*)

RDF 1.1's Literals Section: http://www.w3.org/TR/rdf-concepts/#section-Graph-Literal

Literals are used for values such as strings, numbers, and dates.

A literal in an RDF graph consists of two or three elements:

- a lexical form, being a Unicode string, which SHOULD be in Normal Form C
- a datatype IRI, being an IRI identifying a datatype that determines how the lexical form maps to a literal value, and
- if and only if the datatype IRI is `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`, a non-empty language tag. The language tag MUST be well-formed according to section 2.2.9 of Tags for identifying languages.

A literal is a language-tagged string if the third element is present. Lexical representations of language tags MAY be converted to lower case. The value space of language tags is always in lower case.

—

For valid XSD datatypes, the lexical form is optionally normalized at construction time. Default behaviour is set by rdflib.NORMALIZE_LITERALS and can be overridden by the normalize parameter to __new__

Equality and hashing of Literals are done based on the lexical form, i.e.:

```
>>> from rdflib.namespace import XSD
```

```
>>> Literal('01') != Literal('1')  # clear - strings differ
True
```

but with data-type they get normalized:

```
>>> Literal('01', datatype=XSD.integer) != Literal('1', datatype=XSD.integer)
False
```

unless disabled:

```
>>> Literal('01', datatype=XSD.integer, normalize=False) != Literal('1',␣
↪datatype=XSD.integer)
True
```

Value based comparison is possible:

```
>>> Literal('01', datatype=XSD.integer).eq(Literal('1', datatype=XSD.float))
True
```

The eq method also provides limited support for basic python types:

```
>>> Literal(1).eq(1) # fine - int compatible with xsd:integer
True
>>> Literal('a').eq('b') # fine - str compatible with plain-lit
False
>>> Literal('a', datatype=XSD.string).eq('a') # fine - str compatible with␣
↪xsd:string
True
>>> Literal('a').eq(1) # not fine, int incompatible with plain-lit
NotImplemented
```

Greater-than/less-than ordering comparisons are also done in value space, when compatible datatypes are used. Incompatible datatypes are ordered by DT, or by lang-tag. For other nodes the ordering is None < BNode < URIRef < Literal

Any comparison with non-rdflib Node are “NotImplemented” In PY3 this is an error.

```
>>> from rdflib import Literal, XSD
>>> lit2006 = Literal('2006-01-01',datatype=XSD.date)
>>> lit2006.toPython()
datetime.date(2006, 1, 1)
>>> lit2006 < Literal('2007-01-01',datatype=XSD.date)
True
>>> Literal(datetime.utcnow()).datatype
```

(continues on next page)

(continued from previous page)

```
rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#dateTime')
>>> Literal(1) > Literal(2) # by value
False
>>> Literal(1) > Literal(2.0) # by value
False
>>> Literal('1') > Literal(1) # by DT
True
>>> Literal('1') < Literal('1') # by lexical form
False
>>> Literal('a', lang='en') > Literal('a', lang='fr') # by lang-tag
False
>>> Literal(1) > URIRef('foo') # by node-type
True
```

The > < operators will eat this NotImplemented and throw a TypeError (py3k):

```
>>> Literal(1).__gt__(2.0)
NotImplemented
```

A literal in an RDF graph contains one or two named components.

All literals have a lexical form being a Unicode string, which SHOULD be in Normal Form C.

Plain literals have a lexical form and optionally a language tag as defined by **RFC 3066**, normalized to lowercase. An exception will be raised if illegal language-tags are passed to *rdflib.term.Literal.__new__()*.

Typed literals have a lexical form and a datatype URI being an RDF URI reference.

Note: When using the language tag, care must be taken not to confuse language with locale. The language tag relates only to human language text. Presentational issues should be addressed in end-user applications.

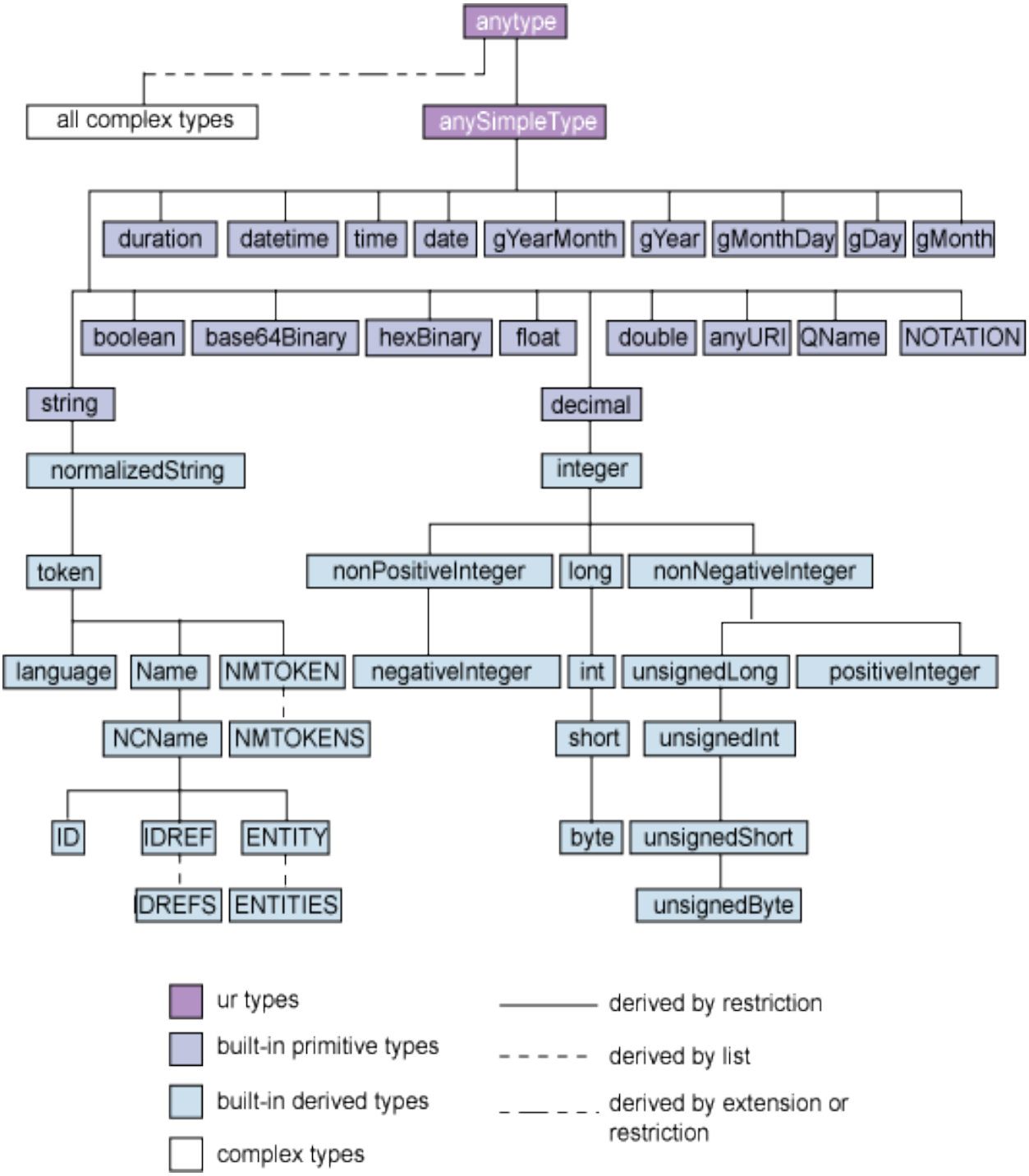
Note: The case normalization of language tags is part of the description of the abstract syntax, and consequently the abstract behaviour of RDF applications. It does not constrain an RDF implementation to actually normalize the case. Crucially, the result of comparing two language tags should not be sensitive to the case of the original input. – RDF Concepts and Abstract Syntax

---

Common XSD datatypes

Most simple literals such as *string* or *integer* have XML Schema (XSD) datatypes defined for them, see the figure below. Additionally, these XSD datatypes are listed in the *XSD Namespace class* that ships with RDFLib, so many Python code editors will prompt you with autocomplete for them when using it.

Remember, you don't *have* to use XSD datatypes and can always make up your own, as GeoSPARQL does, as described above.


anytype
all complex types
anySimpleType
duration
datetime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Python conversions

RDFLib Literals essentially behave like unicode characters with an XML Schema datatype or language attribute.

The class provides a mechanism to both convert Python literals (and their built-ins such as time/date/datetime) into equivalent RDF Literals and (conversely) convert Literals to their Python equivalent. This mapping to and from Python literals is done as follows:

XML Datatype	Python type
None	None[1]
xsd:time	time[2]
xsd:date	date
xsd:dateTime	datetime
xsd:string	None
xsd:normalizedString	None
xsd:token	None
xsd:language	None
xsd:boolean	boolean
xsd:decimal	Decimal
xsd:integer	long
xsd:nonPositiveInteger	int
xsd:long	long
xsd:nonNegativeInteger	int
xsd:negativeInteger	int
xsd:int	long
xsd:unsignedLong	long
xsd:positiveInteger	int
xsd:short	int
xsd:unsignedInt	long
xsd:byte	int
xsd:unsignedShort	int
xsd:unsignedByte	int
xsd:float	float
xsd:double	float
xsd:base64Binary	`base64`
xsd:anyURI	None
rdf:XMLLiteral	`xml.dom.minidom.Document`[3]
rdf:HTML	`xml.dom.minidom.DocumentFragment`

An appropriate data-type and lexical representation can be found using:

`rdflib.term.`**`_castPythonToLiteral`**`(`*obj*, *datatype*`)`

Casts a tuple of a python type and a special datatype URI to a tuple of the lexical value and a datatype URI (or None)

Parameters

- **obj** (`Any`) –
- **datatype** (`Optional[str]`) –

[1] plain literals map directly to value space

[2] Date, time and datetime literals are mapped to Python instances using the isodate package).

[3] this is a bit dirty - by accident the `html5lib` parser produces `DocumentFragments`, and the xml parser `Documents`, letting us use this to decide what datatype when round-tripping.

Return type
Tuple[Any, Optional[str]]

and the other direction with

rdflib.term.**_castLexicalToPython**(*lexical*, *datatype*)

Map a lexical form to the value-space for the given datatype :rtype: Any :returns: a python object for the value or None

Parameters

- **lexical** (Union[str, bytes]) –
- **datatype** (Optional[*URIRef*]) –

All this happens automatically when creating Literal objects by passing Python objects to the constructor, and you never have to do this manually.

You can add custom data-types with *rdflib.term.bind()*, see also *examples.custom_datatype*

2.2 Namespaces and Bindings

RDFLib provides several short-cuts to working with many URIs in the same namespace.

The *rdflib.namespace* defines the *rdflib.namespace.Namespace* class which lets you easily create URIs in a namespace:

```
    from rdflib import Namespace

    EX = Namespace("http://example.org/")
    EX.Person  # a Python attribute for EX. This example is equivalent to rdflib.term.
↪URIRef("http://example.org/Person")

# use dict notation for things that are not valid Python identifiers, e.g.:
    n['first%20name']  # as rdflib.term.URIRef("http://example.org/first%20name")
```

These two styles of namespace creation - object attribute and dict - are equivalent and are made available just to allow for valid RDF namespaces and URIs that are not valid Python identifiers. This isn't just for syntactic things like spaces, as per the example of `first%20name` above, but also for Python reserved words like `class` or `while`, so for the URI `http://example.org/class`, create it with `EX['class']`, not `EX.class`.

2.2.1 Common Namespaces

The `namespace` module defines many common namespaces such as RDF, RDFS, OWL, FOAF, SKOS, PROF, etc. The list of the namespaces provided grows with user contributions to RDFLib.

These Namespaces, and any others that users define, can also be associated with prefixes using the *rdflib.namespace.NamespaceManager*, e.g. using `foaf` for `http://xmlns.com/foaf/0.1/`.

Each RDFLib graph has a *namespace_manager* that keeps a list of namespace to prefix mappings. The namespace manager is populated when reading in RDF, and these prefixes are used when serialising RDF, or when parsing SPARQL queries. Prefixes can be bound with the *rdflib.graph.Graph.bind()* method:

```
from rdflib import Graph, Namespace
from rdflib.namespace import FOAF

```

(continues on next page)

(continued from previous page)

```
EX = Namespace("http://example.org/")

g = Graph()
g.bind("foaf", FOAF)  # bind an RDFLib-provided namespace to a prefix
g.bind("ex", EX)      # bind a user-declared namespace to a prefix
```

The *rdflib.graph.Graph.bind()* method is actually supplied by the *rdflib.namespace.NamespaceManager* class - see next.

2.2.2 NamespaceManager

Each RDFLib graph comes with a *rdflib.namespace.NamespaceManager* instance in the *namespace_manager* field; you can use the *bind()* method of this instance to bind a prefix to a namespace URI, as above, however note that the *NamespaceManager* automatically performs some bindings according to a selected strategy.

Namespace binding strategies are indicated with the `bind_namespaces` input parameter to *NamespaceManager* instances and may be set via `Graph` also:

```
from rdflib import Graph
from rdflib.namespace import NamespaceManager

g = Graph(bind_namespaces="rdflib")  # bind via Graph

g2 = Graph()
nm = NamespaceManager(g2, bind_namespaces="rdflib")  # bind via NamespaceManager
```

Valid strategies are:

- **core:**
  - binds several core RDF prefixes only
  - owl, rdf, rdfs, xsd, xml from the NAMESPACE_PREFIXES_CORE object
  - this is default
- **rdflib:**
  - binds all the namespaces shipped with RDFLib as DefinedNamespace instances
  - all the core namespaces and all the following: brick, csvw, dc, dcat
  - dcmitype, cdterms, dcam, doap, foaf, geo, odrl, org, prof, prov, qb, sdo
  - sh, skos, sosa, ssn, time, vann, void
  - see the NAMESPACE_PREFIXES_RDFLIB object in *rdflib.namespace* for up-to-date list
- **none:**
  - binds no namespaces to prefixes
  - note this is NOT default behaviour
- **cc:**
  - using prefix bindings from prefix.cc which is a online prefixes database
  - not implemented yet - this is aspirational

Re-binding

Note that regardless of the strategy employed, prefixes for namespaces can be overwritten with users preferred prefixes, for example:

```
from rdflib import Graph
from rdflib.namespace import GEO  # imports GeoSPARQL's namespace

g = Graph(bind_namespaces="rdflib")  # binds GeoSPARQL's namespace to prefix 'geo'

g.bind('geosp', GEO, override=True)
```

NamespaceManager also has a method to normalize a given url:

```
from rdflib.namespace import NamespaceManager

nm = NamespaceManager(Graph())
nm.normalizeUri(t)
```

For simple output, or simple serialisation, you often want a nice readable representation of a term. All RDFLib terms have a `.n3()` method, which will return a suitable N3 format and into which you can supply a NamespaceManager instance to provide prefixes, i.e. `.n3(namespace_manager=some_nm)`:

```
>>> from rdflib import Graph, URIRef, Literal, BNode
>>> from rdflib.namespace import FOAF, NamespaceManager

>>> person = URIRef("http://xmlns.com/foaf/0.1/Person")
>>> person.n3()
'<http://xmlns.com/foaf/0.1/Person>'

>>> g = Graph()
>>> g.bind("foaf", FOAF)

>>> person.n3(g.namespace_manager)
'foaf:Person'

>>> l = Literal(2)
>>> l.n3()
'"2"^^<http://www.w3.org/2001/XMLSchema#integer>'

>>> l.n3(NamespaceManager(Graph(), bind_namespaces="core"))
'"2"^^xsd:integer'
```

The namespace manage also has a useful method `compute_qname` `g.namespace_manager.compute_qname(x)` (or just `g.compute_qname(x)`) which takes a URI and decomposes it into the parts:

```
self.assertEqual(g.compute_qname(URIRef("http://foo/bar#baz")),
                 ("ns2", URIRef("http://foo/bar#"), "baz"))
```

2.2.3 Namespaces in SPARQL Queries

The `initNs` argument supplied to *query()* is a dictionary of namespaces to be expanded in the query string. If you pass no `initNs` argument, the namespaces registered with the graphs namespace_manager are used:

```
from rdflib.namespace import FOAF
graph.query('SELECT * WHERE { ?p a foaf:Person }', initNs={'foaf': FOAF})
```

In order to use an empty prefix (e.g. `?a :knows ?b`), use a `PREFIX` directive with no prefix in the SPARQL query to set a default namespace:

```
PREFIX : <http://xmlns.com/foaf/0.1/>
```

2.3 Persistence

RDFLib provides an *abstracted Store API* for persistence of RDF and Notation 3. The *Graph* class works with instances of this API (as the first argument to its constructor) for triple-based management of an RDF store including: garbage collection, transaction management, update, pattern matching, removal, length, and database management (*open()* / *close()* / *destroy()*).

Additional persistence mechanisms can be supported by implementing this API for a different store.

2.3.1 Stores currently shipped with core RDFLib

- *Memory* - not persistent!
- *BerkeleyDB* - on disk persistence via Python's berkeleydb package
- *SPARQLStore* - a read-only wrapper around a remote SPARQL Query endpoint
- *SPARQLUpdateStore* - a read-write wrapper around a remote SPARQL query/update endpoint pair

2.3.2 Usage

In most cases, passing the name of the store to the Graph constructor is enough:

```
from rdflib import Graph

graph = Graph(store='BerkeleyDB')
```

Most stores offering on-disk persistence will need to be opened before reading or writing. When peristing a triplestore, rather than a ConjuntiveGraph quadstore, you need to specify an identifier with which you can open the graph:

```
graph = Graph('BerkeleyDB', identifier='mygraph')

# first time create the store:
graph.open('/home/user/data/myRDFLibStore', create=True)

# work with the graph:
data = """
       PREFIX : <https://example.org/>
```

(continues on next page)

(continued from previous page)

```
        :a :b :c .
        :d :e :f .
        :d :g :h .
        """
graph.parse(data=data, format="ttl")

# when done!
graph.close()
```

When done, *close()* must be called to free the resources associated with the store.

2.3.3 Additional store plugins

More store implementations are available in RDFLib extension projects:

- rdflib-sqlalchemy, which supports stored on a wide-variety of RDBMs backends,
- rdflib-leveldb - a store on to of Google's LevelDB key-value store.
- rdflib-kyotocabinet - a store on to of the Kyoto Cabinet key-value store.

2.3.4 Example

- *examples.berkeleydb_example* contains an example for using a BerkeleyDB store.
- *examples.sparqlstore_example* contains an example for using a SPARQLStore.

2.4 Merging graphs

> Graphs share blank nodes only if they are derived from graphs described by documents or other structures (such as an RDF dataset) that explicitly provide for the sharing of blank nodes between different RDF graphs. Simply downloading a web document does not mean that the blank nodes in a resulting RDF graph are the same as the blank nodes coming from other downloads of the same document or from the same RDF source.

RDF applications which manipulate concrete syntaxes for RDF which use blank node identifiers should take care to keep track of the identity of the blank nodes they identify. Blank node identifiers often have a local scope, so when RDF from different sources is combined, identifiers may have to be changed in order to avoid accidental conflation of distinct blank nodes.

For example, two documents may both use the blank node identifier "_:x" to identify a blank node, but unless these documents are in a shared identifier scope or are derived from a common source, the occurrences of "_:x" in one document will identify a different blank node than the one in the graph described by the other document. When graphs are formed by combining RDF from multiple sources, it may be necessary to standardize apart the blank node identifiers by replacing them by others which do not occur in the other document(s).

(copied directly from https://www.w3.org/TR/rdf11-mt/#shared-blank-nodes-unions-and-merges)

In RDFLib, blank nodes are given unique IDs when parsing, so graph merging can be done by simply reading several files into the same graph:

```
from rdflib import Graph

graph = Graph()

graph.parse(input1)
graph.parse(input2)
```

graph now contains the merged graph of input1 and input2.

Note: However, the set-theoretic graph operations in RDFLib are assumed to be performed in sub-graphs of some larger data-base (for instance, in the context of a *ConjunctiveGraph*) and assume shared blank node IDs, and therefore do NOT do *correct* merging, i.e.:

```
from rdflib import Graph

g1 = Graph()
g1.parse(input1)

g2 = Graph()
g2.parse(input2)

graph = g1 + g2
```

May cause unwanted collisions of blank-nodes in graph.

2.5 Upgrading 5.0.0 to 6.0.0

6.0.0 fully adopts Python 3 practices and drops Python 2 support so it is neater, faster and generally more modern than 5.0.0. It also tidies up the Graph API (removing duplicate functions) so it does include a few breaking changes. Additionally, there is a long list of PRs merged into 6.0.0 adding a number of small fixes and features which are listed below.

RDFLib version 5.0.0 was released in 2020, 3 years after the previous version (4.2.2) and is fundamentally 5.0.0 compatible with. If you need very long-term backwards-compatibility or Python 2 support, you need 5.0.0.

2.5.1 Major Changes

The most notable changes in RDFLib 6.0.0 are:

Python 3.7+

- The oldest version of python you can use to run RDFLib is now 3.7.
- This is a big jump from RDFLib 5.0.0 that worked on python 2.7 and 3.5.
- This change is to allow the library maintainers to adopt more modern development tools, newer language features, and avoid the need to support EOL versions of python in he future

JSON-LD integration and JSON-LD 1.1

- The json-ld serializer/parser plugin was by far the most commonly used RDFLib addon.
- Last year we brought it under the RDFLib org in Github
- Now for 6.0.0 release the JSON-LD serializer and parser are integrated into RDFLib core
- This includes the experimental support for the JSON-LD v1.1 spec
- You no longer need to install the json-ld dependency separately.

2.5.2 All Changes

This list has been assembled from Pull Request and commit information.

General Bugs Fixed:

- Pr 451 redux PR #978

Enhanced Features:

- Register additional serializer plugins for SPARQL mime types. PR #987

SPARQL Fixes:

- Total order patch patch PR #862

Code Quality and Cleanups:

- a slightly opinionated autopep8 run PR #870

Testing:

- 3.7 for travis PR #864

Documentation Fixes:

- Fix a doc string in the query module PR #976

Integrade JSON-LD into RDFLib:

PR #1354

2.6 Upgrading 4.2.2 to 5.0.0

RDFLib version 5.0.0 appeared over 3 years after the previous release, 4.2.2 and contains a large number of both enhancements and bug fixes. Fundamentally though, 5.0.0 is compatible with 4.2.2.

2.6.1 Major Changes

Literal Ordering

Literal total ordering PR #793 is implemented. That means all literals can now be compared to be greater than or less than any other literal. This is required for implementing some specific SPARQL features, but it is counter-intuitive to those who are expecting a TypeError when certain normally-incompatible types are compared. For example, comparing a `Literal(int(1), datatype=xsd:integer)` to `Literal(datetime.date(10,01,2020), datatype=xsd:date)` using a > or < operator in rdflib 4.2.2 and earlier, would normally throw a TypeError, however in rdflib 5.0.0 this operation now returns a True or False according to the Literal Total Ordering according the rules outlined in PR #793

Removed RDF Parsers

The RDFa and Microdata format RDF parsers were removed from rdflib. There are still other python libraries available to implement these parsers.

2.6.2 All Changes

This list has been assembled from Pull Request and commit information.

General Bugs Fixed:

- Pr 451 redux PR #978
- NTriples fails to parse URIs with only a scheme ISSUE #920 PR #974
- cannot clone it on windows - Remove colons from test result files. Fix #901. ISSUE #901 PR #971
- Add requirement for requests to setup.py PR #969
- fixed URIRef including native unicode characters PR #961
- DCTERMS.format not working ISSUE #932
- infixowl.manchesterSyntax do not encode strings PR #906
- Fix blank node label to not contain '_:' during parsing PR #886
- rename new SPARQLWrapper to SPARQLConnector PR #872
- Fix #859. Unquote and Uriquote Literal Datatype. PR #860
- Parsing nquads ISSUE #786
- ntriples spec allows for upper-cased lang tag, fixes #782 PR #784
- Error parsing N-Triple file using RDFlib ISSUE #782
- Adds escaped single quote to literal parser PR #736
- N3 parse error on single quote within single quotes ISSUE #732

- Fixed #725 PR #730
- test for issue #725: canonicalization collapses BNodes PR #726
- RGDA1 graph canonicalization sometimes still collapses distinct BNodes ISSUE #725
- Accept header should use a q parameter PR #720
- Added test for Issue #682 and fixed. PR #718
- Incompatibility with Python3: unichr ISSUE #687
- namespace.py include colon in ALLOWED_NAME_CHARS PR #663
- namespace.py fix compute_qname missing namespaces PR #649
- RDFa parsing Error! `__init__()` got an unexpected keyword argument 'encoding' ISSUE #639
- Bugfix: `term.Literal.__add__` PR #451
- fixup of #443 PR #445
- Microdata to rdf second edition bak PR #444

Enhanced Features:

- Register additional serializer plugins for SPARQL mime types. PR #987
- Pr 388 redux PR #979
- Allows RDF terms introduced by JSON-LD 1.1 PR #970
- make SPARQLConnector work with DBpedia PR #941
- ClosedNamespace returns right exception for way of access PR #866
- Not adding all namespaces for n3 serializer PR #832
- Adds basic support of xsd:duration PR #808
- Add possibility to set authority and basepath to skolemize graph PR #807
- Change notation3 list realization to non-recursive function. PR #805
- Suppress warning for not using custom encoding. PR #800
- Add support to parsing large xml inputs ISSUE #749 PR #750
- improve hash efficiency by directly using str/unicode hash PR #746
- Added the csvw prefix to the RDFa initial context. PR #594
- syncing changes from pyMicrodata PR #587
- Microdata parser: updated the parser to the latest version of the microdata->rdf note (published in December 2014) PR #443
- Literal.toPython() support for xsd:hexBinary PR #388

SPARQL Fixes:

- Total order patch patch PR #862
- use <<= instead of deprecated << PR #861
- Fix #847 PR #856
- RDF Literal "1"^^xsd:boolean should _not_ coerce to True ISSUE #847
- Makes NOW() return an UTC date PR #844
- NOW() SPARQL should return an xsd:dateTime with a timezone ISSUE #843
- fix property paths bug: issue #715 PR #822 ISSUE #715
- MulPath: correct behaviour of n3() PR #820
- Literal total ordering PR #793
- Remove SPARQLWrapper dependency PR #744
- made UNION faster by not preventing duplicates PR #741
- added a hook to add custom functions to SPARQL PR #723
- Issue714 PR #717
- Use <<= instead of deprecated << in SPARQL parser PR #417
- Custom FILTER function for SPARQL engine ISSUE #274

Code Quality and Cleanups:

- a slightly opinionated autopep8 run PR #870
- remove rdfa and microdata parsers from core RDFLib PR #828
- ClosedNamespace KeyError -> AttributeError PR #827
- typo in rdflib/plugins/sparql/update.py ISSUE #760
- Fix logging in interactive mode PR #731
- make namespace module flake8-compliant, change exceptions in that mod… PR #711
- delete ez_setup.py? ISSUE #669
- code duplication issue between rdflib and pymicrodata ISSUE #582
- Transition from 2to3 to use of six.py to be merged in 5.0.0-dev PR #519
- sparqlstore drop deprecated methods and args PR #516
- python3 code seems shockingly inefficient ISSUE #440
- removed md5_term_hash, fixes #240 PR #439 ISSUE #240

Testing:

- 3.7 for travis PR #864
- Added trig unit tests to highlight some current parsing/serializing issues PR #431

Documentation Fixes:

- Fix a doc string in the query module PR #976
- setup.py: Make the license field use an SPDX identifier PR #789
- Update README.md PR #764
- Update namespaces_and_bindings.rst PR #757
- DOC: README.md: rdflib-jsonld, https uris PR #712
- make doctest support py2/py3 ISSUE #707
- `pip install rdflib` (as per README.md) gets OSError on Mint 18.1 ISSUE #704 PR #717
- Use <<= instead of deprecated << in SPARQL parser PR #417
- Custom FILTER function for SPARQL engine ISSUE #274

2.7 Security Considerations

RDFLib is designed to access arbitrary network and file resources, in some cases these are directly requested resources, in other cases they are indirectly referenced resources.

An example of where indirect resources are access is JSON-LD processing, where network or file resources referenced by `@context` values will be loaded and processed.

RDFLib also supports SPARQL, which has federated query capabilities that allow queries to query arbitrary remote endpoints.

If you are using RDFLib to process untrusted documents or queries you should take measures to restrict file and network access.

Some measures that can be taken to restrict file and network access are:

- *Operating System Security Measures*.
- *Python Runtime Audit Hooks*.
- *Custom URL Openers*.

Of these, operating system security measures are recommended. The other measures work, but they are not as effective as operating system security measures, and even if they are used they should be used in conjunction with operating system security measures.

2.7.1 Operating System Security Measures

Most operating systems provide functionality that can be used to restrict network and file access of a process.

Some examples of these include:

- Open Container Initiative (OCI) Containers (aka Docker containers).

  Most OCI runtimes provide mechanisms to restrict network and file access of containers. For example, using Docker, you can limit your container to only being access files explicitly mapped into the container and only access the network through a firewall. For more information refer to the documentation of the tool you use to manage your OCI containers:

  - Kubernetes
  - Docker
  - Podman

- firejail can be used to sandbox a process on Linux and restrict its network and file access.
- File and network access restrictions.

  Most operating systems provide a way to restrict operating system users to only being able to access files and network resources that are explicitly allowed. Applications that process untrusted input could be run as a user with these restrictions in place.

Many other measures are available, however, listing them outside the scope of this document.

Of the listed measures OCI containers are recommended. In most cases, OCI containers are constrained by default and can't access the loopback interface and can only access files that are explicitly mapped into the container.

2.7.2 Python Runtime Audit Hooks

From Python 3.8 onwards, Python provides a mechanism to install runtime audit hooks that can be used to limit access to files and network resources.

The runtime audit hook system is described in more detail in PEP 578 – Python Runtime Audit Hooks.

Runtime audit hooks can be installed using the sys.addaudithook function, and will then get called when audit events occur. The audit events raised by the Python runtime and standard library are described in Python's audit events table.

RDFLib uses `urllib.request.urlopen` for HTTP, HTTPS and other network access, and this function raises a `urllib.Request` audit event. For file access, RDFLib uses `open`, which raises an `open` audit event.

Users of RDFLib can install audit hooks that react to these audit events and raises an exception when an attempt is made to access files or network resources that are not explicitly allowed.

RDFLib's test suite includes tests which verify that audit hooks can block access to network and file resources.

RDFLib also includes an example that shows how runtime audit hooks can be used to restrict network and file access in *`secure_with_audit`*.

2.7.3 Custom URL Openers

RDFLib uses the `urllib.request.urlopen` for HTTP, HTTPS and other network access. This function will use a `urllib.request.OpenerDirector` installed with `urllib.request.install_opener` to open the URLs.

Users of RDFLib can install a custom URL opener that raise an exception when an attempt is made to access network resources that are not explicitly allowed.

RDFLib's test suite includes tests which verify that custom URL openers can be used to block access to network resources.

RDFLib also includes an example that shows how a custom opener can be used to restrict network access in *secure_with_urlopen*.

CHAPTER

THREE

REFERENCE

The nitty-gritty details of everything.

API reference:

3.1 rdflib

3.1.1 rdflib package

Subpackages

rdflib.extras package

Submodules

rdflib.extras.cmdlineutils module

rdflib.extras.cmdlineutils.**main**(*target*, *_help=<function _help>*, *options=''*, *stdin=True*)

A main function for tools that read RDF from files given on commandline or from STDIN (if stdin parameter is true)

rdflib.extras.describer module

A Describer is a stateful utility for creating RDF statements in a semi-declarative manner. It has methods for creating literal values, rel and rev resource relations (somewhat resembling RDFa).

The *Describer.rel* and *Describer.rev* methods return a context manager which sets the current about to the referenced resource for the context scope (for use with the `with` statement).

Full example in the `to_rdf` method below:

```
>>> import datetime
>>> from rdflib.graph import Graph
>>> from rdflib.namespace import Namespace, RDFS, FOAF
>>>
>>> ORG_URI = "http://example.org/"
>>>
>>> CV = Namespace("http://purl.org/captsolo/resume-rdf/0.2/cv#")
>>>
```

(continues on next page)

(continued from previous page)

```
>>> class Person(object):
...     def __init__(self):
...         self.first_name = u"Some"
...         self.last_name = u"Body"
...         self.username = "some1"
...         self.presentation = u"Just a Python & RDF hacker."
...         self.image = "/images/persons/" + self.username + ".jpg"
...         self.site = "http://example.net/"
...         self.start_date = datetime.date(2009, 9, 4)
...     def get_full_name(self):
...         return u" ".join([self.first_name, self.last_name])
...     def get_absolute_url(self):
...         return "/persons/" + self.username
...     def get_thumbnail_url(self):
...         return self.image.replace('.jpg', '-thumb.jpg')
...
...     def to_rdf(self):
...         graph = Graph()
...         graph.bind('foaf', FOAF)
...         graph.bind('cv', CV)
...         lang = 'en'
...         d = Describer(graph, base=ORG_URI)
...         d.about(self.get_absolute_url()+'#person')
...         d.rdftype(FOAF.Person)
...         d.value(FOAF.name, self.get_full_name())
...         d.value(FOAF.givenName, self.first_name)
...         d.value(FOAF.familyName, self.last_name)
...         d.rel(FOAF.homepage, self.site)
...         d.value(RDFS.comment, self.presentation, lang=lang)
...         with d.rel(FOAF.depiction, self.image):
...             d.rdftype(FOAF.Image)
...             d.rel(FOAF.thumbnail, self.get_thumbnail_url())
...         with d.rev(CV.aboutPerson):
...             d.rdftype(CV.CV)
...             with d.rel(CV.hasWorkHistory):
...                 d.value(CV.startDate, self.start_date)
...                 d.rel(CV.employedIn, ORG_URI+"#company")
...         return graph
...
>>> person_graph = Person().to_rdf()
>>> expected = Graph().parse(data='''<?xml version="1.0" encoding="utf-8"?>
... <rdf:RDF
...   xmlns:foaf="http://xmlns.com/foaf/0.1/"
...   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
...   xmlns:cv="http://purl.org/captsolo/resume-rdf/0.2/cv#"
...   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
...   <foaf:Person rdf:about="http://example.org/persons/some1#person">
...     <foaf:name>Some Body</foaf:name>
...     <foaf:givenName>Some</foaf:givenName>
...     <foaf:familyName>Body</foaf:familyName>
...     <foaf:depiction>
...       <foaf:Image
```

(continues on next page)

(continued from previous page)

```
...         rdf:about=
...             "http://example.org/images/persons/some1.jpg">
...         <foaf:thumbnail
...         rdf:resource=
...             "http://example.org/images/persons/some1-thumb.jpg"/>
...       </foaf:Image>
...     </foaf:depiction>
...     <rdfs:comment xml:lang="en">
...             Just a Python &amp;amp; RDF hacker.
...     </rdfs:comment>
...     <foaf:homepage rdf:resource="http://example.net/"/>
...   </foaf:Person>
...   <cv:CV>
...     <cv:aboutPerson
...         rdf:resource="http://example.org/persons/some1#person">
...     </cv:aboutPerson>
...     <cv:hasWorkHistory>
...       <rdf:Description>
...         <cv:startDate
...             rdf:datatype="http://www.w3.org/2001/XMLSchema#date"
...             >2009-09-04</cv:startDate>
...         <cv:employedIn rdf:resource="http://example.org/#company"/>
...       </rdf:Description>
...     </cv:hasWorkHistory>
...   </cv:CV>
... </rdf:RDF>
... ''', format="xml")
>>>
>>> from rdflib.compare import isomorphic
>>> isomorphic(person_graph, expected)
True
```

class rdflib.extras.describer.**Describer**(*graph=None*, *about=None*, *base=None*)

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.extras.describer', '__init__': <function Describer.__init__>, 'about': <function Describer.about>, 'value': <function Describer.value>, 'rel': <function Describer.rel>, 'rev': <function Describer.rev>, 'rdftype': <function Describer.rdftype>, '_current': <function Describer._current>, '_subject_stack': <function Describer._subject_stack>, '__dict__': <attribute '__dict__' of 'Describer' objects>, '__weakref__': <attribute '__weakref__' of 'Describer' objects>, '__doc__': None, '__annotations__': {}})

__init__(*graph=None*, *about=None*, *base=None*)

__module__ = 'rdflib.extras.describer'

__weakref__

list of weak references to the object (if defined)

about(*subject*, ***kws*)

Sets the current subject. Will convert the given object into an URIRef if it's not an Identifier.

Usage:

```
>>> d = Describer()
>>> d._current()
rdflib.term.BNode(...)
>>> d.about("http://example.org/")
>>> d._current()
rdflib.term.URIRef(u'http://example.org/')
```

rdftype(*t*)

Shorthand for setting rdf:type of the current subject.

Usage:

```
>>> from rdflib import URIRef
>>> from rdflib.namespace import RDF, RDFS
>>> d = Describer(about="http://example.org/")
>>> d.rdftype(RDFS.Resource)
>>> (URIRef('http://example.org/'),
...     RDF.type, RDFS.Resource) in d.graph
True
```

rel(*p*, *o=None*, ***kws*)

Set an object for the given property. Will convert the given object into an `URIRef` if it's not an `Identifier`. If none is given, a new `BNode` is used.

Returns a context manager for use in a `with` block, within which the given object is used as current subject.

Usage:

```
>>> from rdflib import URIRef
>>> from rdflib.namespace import RDF, RDFS
>>> d = Describer(about="/", base="http://example.org/")
>>> _ctxt = d.rel(RDFS.seeAlso, "/about")
>>> d.graph.value(URIRef('http://example.org/'), RDFS.seeAlso)
rdflib.term.URIRef(u'http://example.org/about')

>>> with d.rel(RDFS.seeAlso, "/more"):
...     d.value(RDFS.label, "More")
>>> (URIRef('http://example.org/'), RDFS.seeAlso,
...         URIRef('http://example.org/more')) in d.graph
True
>>> d.graph.value(URIRef('http://example.org/more'), RDFS.label)
rdflib.term.Literal(u'More')
```

rev(*p*, *s=None*, ***kws*)

Same as `rel`, but uses current subject as *object* of the relation. The given resource is still used as subject in the returned context manager.

Usage:

```
>>> from rdflib import URIRef
>>> from rdflib.namespace import RDF, RDFS
>>> d = Describer(about="http://example.org/")
>>> with d.rev(RDFS.seeAlso, "http://example.net/"):
...     d.value(RDFS.label, "Net")
>>> (URIRef('http://example.net/'), RDFS.seeAlso,
```

(continues on next page)

(continued from previous page)

```
...         URIRef('http://example.org/')) in d.graph
True
>>> d.graph.value(URIRef('http://example.net/'), RDFS.label)
rdflib.term.Literal(u'Net')
```

value(*p*, *v*, ***kws*)

Set a literal value for the given property. Will cast the value to an `Literal` if a plain literal is given.

Usage:

```
>>> from rdflib import URIRef
>>> from rdflib.namespace import RDF, RDFS
>>> d = Describer(about="http://example.org/")
>>> d.value(RDFS.label, "Example")
>>> d.graph.value(URIRef('http://example.org/'), RDFS.label)
rdflib.term.Literal(u'Example')
```

rdflib.extras.describer.**cast_identifier**(*ref*, ***kws*)

rdflib.extras.describer.**cast_value**(*v*, ***kws*)

rdflib.extras.external_graph_libs module

rdflib.extras.external_graph_libs.**rdflib_to_graphtool**(*graph*, *v_prop_names=['term']*, *e_prop_names=['term']*, *transform_s=<function <lambda>>*, *transform_p=<function <lambda>>*, *transform_o=<function <lambda>>*)

Converts the given graph into a graph_tool.Graph().

The subjects and objects are the later vertices of the Graph. The predicates become edges.

Parameters

- graph: a rdflib.Graph.
- v_prop_names: a list of names for the vertex properties. The default is set to ['term'] (see transform_s, transform_o below).
- e_prop_names: a list of names for the edge properties.
- transform_s: callable with s, p, o input. Should return a dictionary containing a value for each name in v_prop_names. By default is set to {'term': s} which in combination with v_prop_names = ['term'] adds s as 'term' property to the generated vertex for s.
- transform_p: similar to transform_s, but wrt. e_prop_names. By default returns {'term': p} which adds p as a property to the generated edge between the vertex for s and the vertex for o.
- transform_o: similar to transform_s.

Returns: graph_tool.Graph()

```
>>> from rdflib import Graph, URIRef, Literal
>>> g = Graph()
>>> a, b, l = URIRef('a'), URIRef('b'), Literal('l')
```

(continues on next page)

(continued from previous page)

```
>>> p, q = URIRef('p'), URIRef('q')
>>> edges = [(a, p, b), (a, q, b), (b, p, a), (b, p, l)]
>>> for t in edges:
...     g.add(t)
...
>>> mdg = rdflib_to_graphtool(g)
>>> len(list(mdg.edges()))
4
>>> from graph_tool import util as gt_util
>>> vpterm = mdg.vertex_properties['term']
>>> va = gt_util.find_vertex(mdg, vpterm, a)[0]
>>> vb = gt_util.find_vertex(mdg, vpterm, b)[0]
>>> vl = gt_util.find_vertex(mdg, vpterm, l)[0]
>>> (va, vb) in [(e.source(), e.target()) for e in list(mdg.edges())]
True
>>> epterm = mdg.edge_properties['term']
>>> len(list(gt_util.find_edge(mdg, epterm, p))) == 3
True
>>> len(list(gt_util.find_edge(mdg, epterm, q))) == 1
True
```

```
>>> mdg = rdflib_to_graphtool(
...     g,
...     e_prop_names=[str('name')],
...     transform_p=lambda s, p, o: {str('name'): unicode(p)})
>>> epterm = mdg.edge_properties['name']
>>> len(list(gt_util.find_edge(mdg, epterm, unicode(p)))) == 3
True
>>> len(list(gt_util.find_edge(mdg, epterm, unicode(q)))) == 1
True
```

Parameters

- **graph** (*Graph*) –
- **v_prop_names** (List[str]) –
- **e_prop_names** (List[str]) –

rdflib.extras.external_graph_libs.**rdflib_to_networkx_digraph**(*graph*, *calc_weights=True*, *edge_attrs=<function <lambda>>*, ***kwds*)

Converts the given graph into a networkx.DiGraph.

As an rdflib.Graph() can contain multiple edges between nodes, by default adds the a 'triples' attribute to the single DiGraph edge with a list of all triples between s and o. Also by default calculates the edge weight as the length of triples.

Parameters

- `graph`: a rdflib.Graph.
- `calc_weights`: If true calculate multi-graph edge-count as edge 'weight'

- **edge_attrs: Callable to construct later edge_attributes. It receives**
  3 variables (s, p, o) and should construct a dictionary that is passed to networkx's add_edge(s, o, **attrs) function.

  By default this will include setting the 'triples' attribute here, which is treated specially by us to be merged. Other attributes of multi-edges will only contain the attributes of the first edge. If you don't want the 'triples' attribute for tracking, set this to `lambda s, p, o: {}`.

Returns: networkx.DiGraph

```
>>> from rdflib import Graph, URIRef, Literal
>>> g = Graph()
>>> a, b, l = URIRef('a'), URIRef('b'), Literal('l')
>>> p, q = URIRef('p'), URIRef('q')
>>> edges = [(a, p, b), (a, q, b), (b, p, a), (b, p, l)]
>>> for t in edges:
...     g.add(t)
...
>>> dg = rdflib_to_networkx_digraph(g)
>>> dg[a][b]['weight']
2
>>> sorted(dg[a][b]['triples']) == [(a, p, b), (a, q, b)]
True
>>> len(dg.edges())
3
>>> dg.size()
3
>>> dg.size(weight='weight')
4.0
```

```
>>> dg = rdflib_to_networkx_graph(g, False, edge_attrs=lambda s,p,o:{})
>>> 'weight' in dg[a][b]
False
>>> 'triples' in dg[a][b]
False
```

Parameters

- **graph** (*Graph*) –
- **calc_weights** (bool) –

rdflib.extras.external_graph_libs.**rdflib_to_networkx_graph**(*graph*, *calc_weights=True*, *edge_attrs=<function <lambda>>*, ***kwds*)

Converts the given graph into a networkx.Graph.

As an rdflib.Graph() can contain multiple directed edges between nodes, by default adds the a 'triples' attribute to the single DiGraph edge with a list of triples between s and o in graph. Also by default calculates the edge weight as the len(triples).

Parameters

- graph: a rdflib.Graph.
- calc_weights: If true calculate multi-graph edge-count as edge 'weight'

- **edge_attrs: Callable to construct later edge_attributes. It receives**
  3 variables (s, p, o) and should construct a dictionary that is passed to networkx's add_edge(s, o, **attrs) function.

  By default this will include setting the 'triples' attribute here, which is treated specially by us to be merged. Other attributes of multi-edges will only contain the attributes of the first edge. If you don't want the 'triples' attribute for tracking, set this to `lambda s, p, o: {}`.

Returns:
networkx.Graph

```
>>> from rdflib import Graph, URIRef, Literal
>>> g = Graph()
>>> a, b, l = URIRef('a'), URIRef('b'), Literal('l')
>>> p, q = URIRef('p'), URIRef('q')
>>> edges = [(a, p, b), (a, q, b), (b, p, a), (b, p, l)]
>>> for t in edges:
...     g.add(t)
...
>>> ug = rdflib_to_networkx_graph(g)
>>> ug[a][b]['weight']
3
>>> sorted(ug[a][b]['triples']) == [(a, p, b), (a, q, b), (b, p, a)]
True
>>> len(ug.edges())
2
>>> ug.size()
2
>>> ug.size(weight='weight')
4.0
```

```
>>> ug = rdflib_to_networkx_graph(g, False, edge_attrs=lambda s,p,o:{})
>>> 'weight' in ug[a][b]
False
>>> 'triples' in ug[a][b]
False
```

Parameters

- **graph** (*Graph*) –
- **calc_weights** (bool) –

rdflib.extras.external_graph_libs.**rdflib_to_networkx_multidigraph**(*graph*, *edge_attrs=<function <lambda>>*, ***kwds*)

Converts the given graph into a networkx.MultiDiGraph.

The subjects and objects are the later nodes of the MultiDiGraph. The predicates are used as edge keys (to identify multi-edges).

Parameters

- graph: a rdflib.Graph.

- **edge_attrs: Callable to construct later edge_attributes. It receives**
  3 variables (s, p, o) and should construct a dictionary that is passed to networkx's add_edge(s, o, **attrs) function.

  By default this will include setting the MultiDiGraph key=p here. If you don't want to be able to re-identify the edge later on, you can set this to `lambda s, p, o: {}`. In this case MultiDiGraph's default (increasing ints) will be used.

Returns:
networkx.MultiDiGraph

```
>>> from rdflib import Graph, URIRef, Literal
>>> g = Graph()
>>> a, b, l = URIRef('a'), URIRef('b'), Literal('l')
>>> p, q = URIRef('p'), URIRef('q')
>>> edges = [(a, p, b), (a, q, b), (b, p, a), (b, p, l)]
>>> for t in edges:
...     g.add(t)
...
>>> mdg = rdflib_to_networkx_multidigraph(g)
>>> len(mdg.edges())
4
>>> mdg.has_edge(a, b)
True
>>> mdg.has_edge(a, b, key=p)
True
>>> mdg.has_edge(a, b, key=q)
True
```

```
>>> mdg = rdflib_to_networkx_multidigraph(g, edge_attrs=lambda s,p,o: {})
>>> mdg.has_edge(a, b, key=0)
True
>>> mdg.has_edge(a, b, key=1)
True
```

Parameters
graph (*Graph*) –

rdflib.extras.infixowl module

RDFLib Python binding for OWL Abstract Syntax

see: **http://www.w3.org/TR/owl-semantics/syntax.html**
http://owl-workshop.man.ac.uk/acceptedLong/submission_9.pdf

3.2.3 Axioms for complete classes without using owl:equivalentClass

Named class description of type 2 (with owl:oneOf) or type 4-6 (with owl:intersectionOf, owl:unionOf or owl:complementOf

Uses Manchester Syntax for __repr__

```
>>> exNs = Namespace('http://example.com/')
>>> namespace_manager = NamespaceManager(Graph())
```

(continues on next page)

(continued from previous page)

```
>>> namespace_manager.bind('ex', exNs, override=False)
>>> namespace_manager.bind('owl', OWL, override=False)
>>> g = Graph()
>>> g.namespace_manager = namespace_manager
```

Now we have an empty graph, we can construct OWL classes in it using the Python classes defined in this module

```
>>> a = Class(exNs.Opera, graph=g)
```

Now we can assert rdfs:subClassOf and owl:equivalentClass relationships (in the underlying graph) with other classes using the 'subClassOf' and 'equivalentClass' descriptors which can be set to a list of objects for the corresponding predicates.

```
>>> a.subClassOf = [exNs.MusicalWork]
```

We can then access the rdfs:subClassOf relationships

```
>>> print(list(a.subClassOf))
[Class: ex:MusicalWork ]
```

This can also be used against already populated graphs:

```
>>> owlGraph = Graph().parse(str(OWL))
>>> namespace_manager.bind('owl', OWL, override=False)
>>> owlGraph.namespace_manager = namespace_manager
>>> list(Class(OWL.Class, graph=owlGraph).subClassOf)
[Class: rdfs:Class ]
```

Operators are also available. For instance we can add ex:Opera to the extension of the ex:CreativeWork class via the '+=' operator

```
>>> a
Class: ex:Opera SubClassOf: ex:MusicalWork
>>> b = Class(exNs.CreativeWork, graph=g)
>>> b += a
>>> print(sorted(a.subClassOf, key=lambda c:c.identifier))
[Class: ex:CreativeWork , Class: ex:MusicalWork ]
```

And we can then remove it from the extension as well

```
>>> b -= a
>>> a
Class: ex:Opera SubClassOf: ex:MusicalWork
```

Boolean class constructions can also be created with Python operators. For example, The | operator can be used to construct a class consisting of a owl:unionOf the operands:

```
>>> c =  a | b | Class(exNs.Work, graph=g)
>>> c
( ex:Opera OR ex:CreativeWork OR ex:Work )
```

Boolean class expressions can also be operated as lists (using python list operators)

```
>>> del c[c.index(Class(exNs.Work, graph=g))]
>>> c
( ex:Opera OR ex:CreativeWork )
```

The '&' operator can be used to construct class intersection:

```
>>> woman = Class(exNs.Female, graph=g) & Class(exNs.Human, graph=g)
>>> woman.identifier = exNs.Woman
>>> woman
( ex:Female AND ex:Human )
>>> len(woman)
2
```

Enumerated classes can also be manipulated

```
>>> contList = [Class(exNs.Africa, graph=g), Class(exNs.NorthAmerica, graph=g)]
>>> EnumeratedClass(members=contList, graph=g)
{ ex:Africa ex:NorthAmerica }
```

owl:Restrictions can also be instantiated:

```
>>> Restriction(exNs.hasParent, graph=g, allValuesFrom=exNs.Human)
( ex:hasParent ONLY ex:Human )
```

Restrictions can also be created using Manchester OWL syntax in 'colloquial' Python >>> exNs.hasParent << some >> Class(exNs.Physician, graph=g) (ex:hasParent SOME ex:Physician)

```
>>> Property(exNs.hasParent, graph=g) << max >> Literal(1)
( ex:hasParent MAX 1 )
```

```
>>> print(g.serialize(format='pretty-xml'))
```

rdflib.extras.infixowl.**AllClasses**(*graph*)

rdflib.extras.infixowl.**AllDifferent**(*members*)

TODO: implement this function

DisjointClasses(' description description { description } ')'

rdflib.extras.infixowl.**AllProperties**(*graph*)

class rdflib.extras.infixowl.**AnnotatableTerms**(*identifier*, *graph=None*, *nameAnnotation=None*, *nameIsLabel=False*)

Bases: *Individual*

Terms in an OWL ontology with rdfs:label and rdfs:comment

Interface with ATTEMPTO (http://attempto.ifi.uzh.ch/site)

Verbalisation of OWL entity IRIS

How are OWL entity IRIs verbalized?

The OWL verbalizer maps OWL entity IRIs to ACE content words such that

- OWL individuals map to ACE proper names (PN)
- OWL classes map to ACE common nouns (CN)

- OWL properties map to ACE transitive verbs (TV)

There are 6 morphological categories that determine the surface form of an IRI:

- singular form of a proper name (e.g. John)
- singular form of a common noun (e.g. man)
- plural form of a common noun (e.g. men)
- singular form of a transitive verb (e.g. mans)
- plural form of a transitive verb (e.g. man)
- past participle form a transitive verb (e.g. manned)

The user has full control over the eventual surface forms of the IRIs but has to choose them in terms of the above categories. Furthermore,

- the surface forms must be legal ACE content words (e.g. they should not contain punctuation symbols);
- the mapping of IRIs to surface forms must be bidirectional within the same word class, in order to be able to (if needed) parse the verbalization back into OWL in a semantics preserving way.

Using the lexicon

It is possible to specify the mapping of IRIs to surface forms using the following annotation properties:

```
http://attempto.ifi.uzh.ch/ace_lexicon#PN_sg
http://attempto.ifi.uzh.ch/ace_lexicon#CN_sg
http://attempto.ifi.uzh.ch/ace_lexicon#CN_pl
http://attempto.ifi.uzh.ch/ace_lexicon#TV_sg
http://attempto.ifi.uzh.ch/ace_lexicon#TV_pl
http://attempto.ifi.uzh.ch/ace_lexicon#TV_vbg
```

For example, the following axioms state that if the IRI "#man" is used as a plural common noun, then the wordform men must be used by the verbalizer. If, however, it is used as a singular transitive verb, then mans must be used.

```
<AnnotationAssertion>
    <AnnotationProperty IRI="http://attempto.ifi.uzh.ch/ace_lexicon#CN_pl"/>
    <IRI>#man</IRI>
    <Literal datatypeIRI="&xsd;string">men</Literal>
</AnnotationAssertion>

<AnnotationAssertion>
    <AnnotationProperty IRI="http://attempto.ifi.uzh.ch/ace_lexicon#TV_sg"/>
    <IRI>#man</IRI>
    <Literal datatypeIRI="&xsd;string">mans</Literal>
</AnnotationAssertion>
```

__init__(*identifier*, *graph=None*, *nameAnnotation=None*, *nameIsLabel=False*)

__module__ = 'rdflib.extras.infixowl'

property comment

handleAnnotation(*val*)

property label

property seeAlso

setupACEAnnotations()

class rdflib.extras.infixowl.**BooleanClass**(*identifier=None*, *operator=rdflib.term.URIRef('http://www.w3.org/2002/07/owl#intersectionOf')*, *members=None*, *graph=None*)

Bases: *OWLRDFListProxy*, *Class*

See: http://www.w3.org/TR/owl-ref/#Boolean

owl:complementOf is an attribute of Class, however

__init__(*identifier=None*, *operator=rdflib.term.URIRef('http://www.w3.org/2002/07/owl#intersectionOf')*, *members=None*, *graph=None*)

__module__ = 'rdflib.extras.infixowl'

__or__(*other*)

Adds other to the list and returns self

__repr__()

Returns the Manchester Syntax equivalent for this class

changeOperator(*newOperator*)

Converts a unionOf / intersectionOf class expression into one that instead uses the given operator

```
>>> testGraph = Graph()
>>> Individual.factoryGraph = testGraph
>>> EX = Namespace("http://example.com/")
>>> namespace_manager = NamespaceManager(Graph())
>>> namespace_manager.bind('ex', EX, override=False)
>>> testGraph.namespace_manager = namespace_manager
>>> fire = Class(EX.Fire)
>>> water = Class(EX.Water)
>>> testClass = BooleanClass(members=[fire,water])
>>> testClass
( ex:Fire AND ex:Water )
>>> testClass.changeOperator(OWL.unionOf)
>>> testClass
( ex:Fire OR ex:Water )
>>> try:
...     testClass.changeOperator(OWL.unionOf)
... except Exception as e:
...     print(e)
The new operator is already being used!
```

copy()

Create a copy of this class

getIntersections = <rdflib.extras.infixowl.Callable object>

getUnions = <rdflib.extras.infixowl.Callable object>

isPrimitive()

serialize(*graph*)

class rdflib.extras.infixowl.**Callable**(*anycallable*)

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.extras.infixowl', '__init__': <function Callable.__init__>, '__dict__': <attribute '__dict__' of 'Callable' objects>, '__weakref__': <attribute '__weakref__' of 'Callable' objects>, '__doc__': None, '__annotations__': {}})

__init__(*anycallable*)

__module__ = 'rdflib.extras.infixowl'

__weakref__

list of weak references to the object (if defined)

rdflib.extras.infixowl.**CastClass**(*c*, *graph=None*)

class rdflib.extras.infixowl.**Class**(*identifier=None*, *subClassOf=None*, *equivalentClass=None*, *disjointWith=None*, *complementOf=None*, *graph=None*, *skipOWLClassMembership=False*, *comment=None*, *nounAnnotations=None*, *nameAnnotation=None*, *nameIsLabel=False*)

Bases: *AnnotatableTerms*

'General form' for classes:

The Manchester Syntax (supported in Protege) is used as the basis for the form of this class

See: http://owl-workshop.man.ac.uk/acceptedLong/submission_9.pdf:

[Annotation] 'Class:' classID {Annotation (('SubClassOf:' ClassExpression) | ('EquivalentTo' ClassExpression) | ('DisjointWith' ClassExpression)) }

Appropriate excerpts from OWL Reference:

".. Subclass axioms provide us with partial definitions: they represent
necessary but not sufficient conditions for establishing class membership of an individual."

".. A class axiom may contain (multiple) owl:equivalentClass statements"

"..A class axiom may also contain (multiple) owl:disjointWith statements.."

"..An owl:complementOf property links a class to precisely one class
description."

__and__(*other*)

Construct an anonymous class description consisting of the intersection of this class and 'other' and return it

```
>>> exNs = Namespace('http://example.com/')
>>> namespace_manager = NamespaceManager(Graph())
>>> namespace_manager.bind('ex', exNs, override=False)
>>> namespace_manager.bind('owl', OWL, override=False)
>>> g = Graph()
>>> g.namespace_manager = namespace_manager
```

Chaining 3 intersections

```
>>> female = Class(exNs.Female, graph=g)
>>> human = Class(exNs.Human, graph=g)
```

(continues on next page)

(continued from previous page)

```
>>> youngPerson = Class(exNs.YoungPerson, graph=g)
>>> youngWoman = female & human & youngPerson
>>> youngWoman
ex:YoungPerson THAT ( ex:Female AND ex:Human )
>>> isinstance(youngWoman, BooleanClass)
True
>>> isinstance(youngWoman.identifier, BNode)
True
```

__eq__(*other*)

Return self==value.

__hash__()

```
>>> b = Class(OWL.Restriction)
>>> c = Class(OWL.Restriction)
>>> len(set([b,c]))
1
```

__iadd__(*other*)

__init__(*identifier=None*, *subClassOf=None*, *equivalentClass=None*, *disjointWith=None*, *complementOf=None*, *graph=None*, *skipOWLClassMembership=False*, *comment=None*, *nounAnnotations=None*, *nameAnnotation=None*, *nameIsLabel=False*)

__invert__()

Shorthand for Manchester syntax's not operator

__isub__(*other*)

__module__ = 'rdflib.extras.infixowl'

__or__(*other*)

Construct an anonymous class description consisting of the union of this class and 'other' and return it

__repr__(*full=False*, *normalization=True*)

Returns the Manchester Syntax equivalent for this class

property annotation

property complementOf

property disjointWith

property equivalentClass

property extent

property extentQuery

isPrimitive()

property parents

computed attributes that returns a generator over taxonomic 'parents' by disjunction, conjunction, and subsumption

```
>>> from rdflib.util import first
>>> exNs = Namespace('http://example.com/')
>>> namespace_manager = NamespaceManager(Graph())
>>> namespace_manager.bind('ex', exNs, override=False)
>>> namespace_manager.bind('owl', OWL, override=False)
>>> g = Graph()
>>> g.namespace_manager = namespace_manager
>>> Individual.factoryGraph = g
>>> brother = Class(exNs.Brother)
>>> sister = Class(exNs.Sister)
>>> sibling = brother | sister
>>> sibling.identifier = exNs.Sibling
>>> sibling
( ex:Brother OR ex:Sister )
>>> first(brother.parents)
Class: ex:Sibling EquivalentTo: ( ex:Brother OR ex:Sister )
>>> parent = Class(exNs.Parent)
>>> male = Class(exNs.Male)
>>> father = parent & male
>>> father.identifier = exNs.Father
>>> list(father.parents)
[Class: ex:Parent , Class: ex:Male ]
```

serialize(*graph*)

setupNounAnnotations(*noun_annotations*)

property subClassOf

subSumpteeIds()

class rdflib.extras.infixowl.**ClassNamespaceFactory**(*value: Union[str, bytes]*)

Bases: *Namespace*

__getattr__(*name*)

__getitem__(*key*, *default=None*)

Return self[key].

__module__ = 'rdflib.extras.infixowl'

term(*name*)

rdflib.extras.infixowl.**CommonNSBindings**(*graph*, *additionalNS=None*)

Takes a graph and binds the common namespaces (rdf,rdfs, & owl)

rdflib.extras.infixowl.**ComponentTerms**(*cls*)

Takes a Class instance and returns a generator over the classes that are involved in its definition, ignoring unnamed classes

rdflib.extras.infixowl.**DeepClassClear**(*class_to_prune*)

Recursively clear the given class, continuing where any related class is an anonymous class

```
>>> EX = Namespace('http://example.com/')
>>> namespace_manager = NamespaceManager(Graph())
```

(continues on next page)

(continued from previous page)

```
>>> namespace_manager.bind('ex', EX, override=False)
>>> namespace_manager.bind('owl', OWL, override=False)
>>> g = Graph()
>>> g.namespace_manager = namespace_manager
>>> Individual.factoryGraph = g
>>> classB = Class(EX.B)
>>> classC = Class(EX.C)
>>> classD = Class(EX.D)
>>> classE = Class(EX.E)
>>> classF = Class(EX.F)
>>> anonClass = EX.someProp << some >> classD
>>> classF += anonClass
>>> list(anonClass.subClassOf)
[Class: ex:F ]
>>> classA = classE | classF | anonClass
>>> classB += classA
>>> classA.equivalentClass = [Class()]
>>> classB.subClassOf = [EX.someProp << some >> classC]
>>> classA
( ex:E OR ex:F OR ( ex:someProp SOME ex:D ) )
>>> DeepClassClear(classA)
>>> classA
(  )
>>> list(anonClass.subClassOf)
[]
>>> classB
Class: ex:B SubClassOf: ( ex:someProp SOME ex:C )
```

```
>>> otherClass = classD | anonClass
>>> otherClass
( ex:D OR ( ex:someProp SOME ex:D ) )
>>> DeepClassClear(otherClass)
>>> otherClass
(  )
>>> otherClass.delete()
>>> list(g.triples((otherClass.identifier, None, None)))
[]
```

class rdflib.extras.infixowl.**EnumeratedClass**(*identifier=None*, *members=None*, *graph=None*)

Bases: *OWLRDFListProxy*, *Class*

Class for owl:oneOf forms:

OWL Abstract Syntax is used

axiom ::= 'EnumeratedClass('
classID ['Deprecated'] { annotation } { individualID } ')'

```
>>> exNs = Namespace('http://example.com/')
>>> namespace_manager = NamespaceManager(Graph())
>>> namespace_manager.bind('ex', exNs, override=False)
>>> namespace_manager.bind('owl', OWL, override=False)
>>> g = Graph()
```

(continues on next page)

(continued from previous page)

```
>>> g.namespace_manager = namespace_manager
>>> Individual.factoryGraph = g
>>> ogbujiBros = EnumeratedClass(exNs.ogbujicBros,
...                              members=[exNs.chime,
...                                       exNs.uche,
...                                       exNs.ejike])
>>> ogbujiBros
{ ex:chime ex:uche ex:ejike }
>>> col = Collection(g, first(
...    g.objects(predicate=OWL.oneOf, subject=ogbujiBros.identifier)))
>>> sorted([g.qname(item) for item in col])
['ex:chime', 'ex:ejike', 'ex:uche']
>>> print(g.serialize(format='n3'))
@prefix ex: <http://example.com/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:ogbujicBros a owl:Class;
    owl:oneOf ( ex:chime ex:uche ex:ejike ) .
```

__init__(*identifier=None*, *members=None*, *graph=None*)

__module__ = 'rdflib.extras.infixowl'

__repr__()

Returns the Manchester Syntax equivalent for this class

isPrimitive()

serialize(*graph*)

rdflib.extras.infixowl.**GetIdentifiedClasses**(*graph*)

class rdflib.extras.infixowl.**Individual**(*identifier=None*, *graph=None*)

Bases: object

A typed individual

__dict__ = mappingproxy({'__module__': 'rdflib.extras.infixowl', '__doc__': '\n A typed individual\n ', 'factoryGraph': <Graph identifier=N78889c59647c434796a6fcca33b012f4 (<class 'rdflib.graph.Graph'>)>, 'serialize': <function Individual.serialize>, '__init__': <function Individual.__init__>, 'clearInDegree': <function Individual.clearInDegree>, 'clearOutDegree': <function Individual.clearOutDegree>, 'delete': <function Individual.delete>, 'replace': <function Individual.replace>, '_get_type': <function Individual._get_type>, '_set_type': <function Individual._set_type>, '_delete_type': <function TermDeletionHelper.__call__.<locals>._remover>, 'type': <property object>, '_get_identifier': <function Individual._get_identifier>, '_set_identifier': <function Individual._set_identifier>, 'identifier': <property object>, '_get_sameAs': <function Individual._get_sameAs>, '_set_sameAs': <function Individual._set_sameAs>, '_delete_sameAs': <function TermDeletionHelper.__call__.<locals>._remover>, 'sameAs': <property object>, '__dict__': <attribute '__dict__' of 'Individual' objects>, '__weakref__': <attribute '__weakref__' of 'Individual' objects>, '__annotations__': {}})

__init__(*identifier=None*, *graph=None*)

__module__ = 'rdflib.extras.infixowl'

__weakref__

list of weak references to the object (if defined)

clearInDegree()

clearOutDegree()

delete()

factoryGraph = <Graph identifier=N78889c59647c434796a6fcca33b012f4 (<class 'rdflib.graph.Graph'>)>

property identifier

replace(*other*)

property sameAs

serialize(*graph*)

property type

class rdflib.extras.infixowl.Infix(*function*)

Bases: object

__call__(*value1*, *value2*)

Call self as a function.

__dict__ = mappingproxy({'__module__': 'rdflib.extras.infixowl', '__init__': <function Infix.__init__>, '__rlshift__': <function Infix.__rlshift__>, '__rshift__': <function Infix.__rshift__>, '__rmatmul__': <function Infix.__rmatmul__>, '__matmul__': <function Infix.__matmul__>, '__call__': <function Infix.__call__>, '__dict__': <attribute '__dict__' of 'Infix' objects>, '__weakref__': <attribute '__weakref__' of 'Infix' objects>, '__doc__': None, '__annotations__': {}})

__init__(*function*)

__matmul__(*other*)

__module__ = 'rdflib.extras.infixowl'

__rlshift__(*other*)

__rmatmul__(*other*)

__rshift__(*other*)

__weakref__

list of weak references to the object (if defined)

exception rdflib.extras.infixowl.MalformedClass

Bases: ValueError

Deprecated since version TODO-NEXT-VERSION: This class will be removed in version 7.0.0.

__module__ = 'rdflib.extras.infixowl'

__weakref__

list of weak references to the object (if defined)

exception rdflib.extras.infixowl.MalformedClassError(*msg*)

Bases: *MalformedClass*

__init__(*msg*)

__module__ = 'rdflib.extras.infixowl'

__repr__()

Return repr(self).

class rdflib.extras.infixowl.OWLRDFListProxy(*rdf_list*, *members=None*, *graph=None*)

Bases: object

__contains__(*item*)

__delitem__(*key*)

__dict__ = mappingproxy({'__module__': 'rdflib.extras.infixowl', '__init__': <function OWLRDFListProxy.__init__>, '__eq__': <function OWLRDFListProxy.__eq__>, '__len__': <function OWLRDFListProxy.__len__>, 'index': <function OWLRDFListProxy.index>, '__getitem__': <function OWLRDFListProxy.__getitem__>, '__setitem__': <function OWLRDFListProxy.__setitem__>, '__delitem__': <function OWLRDFListProxy.__delitem__>, 'clear': <function OWLRDFListProxy.clear>, '__iter__': <function OWLRDFListProxy.__iter__>, '__contains__': <function OWLRDFListProxy.__contains__>, 'append': <function OWLRDFListProxy.append>, '__iadd__': <function OWLRDFListProxy.__iadd__>, '__dict__': <attribute '__dict__' of 'OWLRDFListProxy' objects>, '__weakref__': <attribute '__weakref__' of 'OWLRDFListProxy' objects>, '__doc__': None, '__hash__': None, '__annotations__': {}})

__eq__(*other*)

Equivalence of boolean class constructors is determined by equivalence of its members

__getitem__(*key*)

__hash__ = None

__iadd__(*other*)

__init__(*rdf_list*, *members=None*, *graph=None*)

__iter__()

__len__()

__module__ = 'rdflib.extras.infixowl'

__setitem__(*key*, *value*)

__weakref__

list of weak references to the object (if defined)

append(*item*)

clear()

index(*item*)

class rdflib.extras.infixowl.**Ontology**(*identifier=None*, *imports=None*, *comment=None*, *graph=None*)

Bases: *AnnotatableTerms*

The owl ontology metadata

__init__(*identifier=None*, *imports=None*, *comment=None*, *graph=None*)

__module__ = 'rdflib.extras.infixowl'

property imports

setVersion(*version*)

class rdflib.extras.infixowl.**Property**(*identifier=None*, *graph=None*, *baseType=rdflib.term.URIRef('http://www.w3.org/2002/07/owl#ObjectProperty')*, *subPropertyOf=None*, *domain=None*, *range=None*, *inverseOf=None*, *otherType=None*, *equivalentProperty=None*, *comment=None*, *verbAnnotations=None*, *nameAnnotation=None*, *nameIsLabel=False*)

Bases: *AnnotatableTerms*

axiom ::= 'DatatypeProperty(' datavaluedPropertyID ['Deprecated']
{ annotation } { 'super(' datavaluedPropertyID ')' } ['Functional'] { 'domain(' description ')' } { 'range(' dataRange ')' } ')' | 'ObjectProperty(' individualvaluedPropertyID ['Deprecated'] { annotation } { 'super(' individualvaluedPropertyID ')' } ['inverseOf(' individualvaluedPropertyID ')'] ['Symmetric'] ['Functional' | 'InverseFunctional' | 'Functional' 'InverseFunctional' | 'Transitive'] { 'domain(' description ')' } { 'range(' description ')' } ')

__init__(*identifier=None*, *graph=None*, *baseType=rdflib.term.URIRef('http://www.w3.org/2002/07/owl#ObjectProperty')*, *subPropertyOf=None*, *domain=None*, *range=None*, *inverseOf=None*, *otherType=None*, *equivalentProperty=None*, *comment=None*, *verbAnnotations=None*, *nameAnnotation=None*, *nameIsLabel=False*)

__module__ = 'rdflib.extras.infixowl'

__repr__()
Return repr(self).

property domain

property extent

property inverseOf

property range

replace(*other*)

serialize(*graph*)

setupVerbAnnotations(*verb_annotations*)
OWL properties map to ACE transitive verbs (TV)

There are 6 morphological categories that determine the surface form of an IRI:

singular form of a transitive verb (e.g. mans) plural form of a transitive verb (e.g. man) past participle form a transitive verb (e.g. manned)

http://attempto.ifi.uzh.ch/ace_lexicon#TV_sg http://attempto.ifi.uzh.ch/ace_lexicon#TV_pl http://attempto.ifi.uzh.ch/ace_lexicon#TV_vbg

property subPropertyOf

class rdflib.extras.infixowl.**Restriction**(*onProperty*, *graph=None*, *allValuesFrom=None*, *someValuesFrom=None*, *value=None*, *cardinality=None*, *maxCardinality=None*, *minCardinality=None*, *identifier=None*)

Bases: *Class*

restriction ::= 'restriction(' datavaluedPropertyID dataRestrictionComponent { dataRestrictionComponent } ')' | 'restriction(' individualvaluedPropertyID individualRestrictionComponent { individualRestrictionComponent } ')'

__eq__(*other*)

Equivalence of restrictions is determined by equivalence of the property in question and the restriction 'range'

__hash__()

```
>>> b = Class(OWL.Restriction)
>>> c = Class(OWL.Restriction)
>>> len(set([b,c]))
1
```

__init__(*onProperty*, *graph=None*, *allValuesFrom=None*, *someValuesFrom=None*, *value=None*, *cardinality=None*, *maxCardinality=None*, *minCardinality=None*, *identifier=None*)

__module__ = 'rdflib.extras.infixowl'

__repr__()

Returns the Manchester Syntax equivalent for this restriction

property allValuesFrom

property cardinality

property hasValue

isPrimitive()

property maxCardinality

property minCardinality

property onProperty

restrictionKind()

restrictionKinds =
[rdflib.term.URIRef('http://www.w3.org/2002/07/owl#allValuesFrom'),
rdflib.term.URIRef('http://www.w3.org/2002/07/owl#someValuesFrom'),
rdflib.term.URIRef('http://www.w3.org/2002/07/owl#hasValue'),
rdflib.term.URIRef('http://www.w3.org/2002/07/owl#maxCardinality'),
rdflib.term.URIRef('http://www.w3.org/2002/07/owl#minCardinality')]

serialize(*graph*)

```
>>> g1 = Graph()
>>> g2 = Graph()
>>> EX = Namespace("http://example.com/")
>>> namespace_manager = NamespaceManager(g1)
>>> namespace_manager.bind('ex', EX, override=False)
>>> namespace_manager = NamespaceManager(g2)
>>> namespace_manager.bind('ex', EX, override=False)
>>> Individual.factoryGraph = g1
>>> prop = Property(EX.someProp, baseType=OWL.DatatypeProperty)
>>> restr1 = (Property(
...    EX.someProp,
...    baseType=OWL.DatatypeProperty)) << some >> (Class(EX.Foo))
>>> restr1
( ex:someProp SOME ex:Foo )
>>> restr1.serialize(g2)
>>> Individual.factoryGraph = g2
>>> list(Property(
...     EX.someProp,baseType=None).type
... )
[rdflib.term.URIRef(
    'http://www.w3.org/2002/07/owl#DatatypeProperty')]
```

property someValuesFrom

rdflib.extras.infixowl.**classOrIdentifier**(*thing*)

rdflib.extras.infixowl.**classOrTerm**(*thing*)

rdflib.extras.infixowl.**generateQName**(*graph*, *uri*)

rdflib.extras.infixowl.**manchesterSyntax**(*thing*, *store*, *boolean=None*, *transientList=False*)

Core serialization thing is a Class and is processed as a subject store is an RDFLib Graph to be queried about thing

rdflib.extras.infixowl.**propertyOrIdentifier**(*thing*)

Module contents

rdflib.namespace package

Module contents

Namespace Utilities

RDFLib provides mechanisms for managing Namespaces.

In particular, there is a *Namespace* class that takes as its argument the base URI of the namespace.

```
>>> from rdflib.namespace import Namespace
>>> RDFS = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")
```

Fully qualified URIs in the namespace can be constructed either by attribute or by dictionary access on Namespace instances:

```
>>> RDFS.seeAlso
rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#seeAlso')
>>> RDFS['seeAlso']
rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#seeAlso')
```

Automatic handling of unknown predicates

As a programming convenience, a namespace binding is automatically created when *rdflib.term.URIRef* predicates are added to the graph.

Importable namespaces

The following namespaces are available by directly importing from rdflib:

- BRICK
- CSVW
- DC
- DCAT
- DCMITYPE
- DCTERMS
- DCAM
- DOAP
- FOAF
- ODRL2
- ORG
- OWL
- PROF
- PROV
- QB
- RDF
- RDFS
- SDO
- SH
- SKOS
- SOSA
- SSN
- TIME
- VANN

- VOID
- WGS
- XSD

```
>>> from rdflib.namespace import RDFS
>>> RDFS.seeAlso
rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#seeAlso')
```

class rdflib.namespace.**ClosedNamespace**(*uri: str*, *terms: List[str]*)

Bases: *Namespace*

A namespace with a closed list of members

Trying to create terms not listed is an error

__annotations__ = {'_ClosedNamespace__uris': typing.Dict[str, rdflib.term.URIRef]}

__contains__(*ref*)

Allows to check if a URI is within (starts with) this Namespace.

```
>>> from rdflib import URIRef
>>> namespace = Namespace('http://example.org/')
>>> uri = URIRef('http://example.org/foo')
>>> uri in namespace
True
>>> person_class = namespace['Person']
>>> person_class in namespace
True
>>> obj = URIRef('http://not.example.org/bar')
>>> obj in namespace
False
```

Parameters
ref (str) –

Return type
bool

__dir__()

Default dir() implementation.

Return type
List[str]

__getattr__(*name*)

Parameters
name (str) –

Return type
URIRef

__getitem__(*key*)

Return self[key].

Parameters
key (str) –

Return type
URIRef

__module__ = 'rdflib.namespace'

static __new__(*cls*, *uri*, *terms*)

Parameters

- **uri** (str) –
- **terms** (List[str]) –

__repr__()

Return repr(self).

Return type
str

term(*name*)

Parameters
name (str) –

Return type
URIRef

property uri: str

Return type
str

class rdflib.namespace.**DefinedNamespace**

Bases: object

A Namespace with an enumerated list of members. Warnings are emitted if unknown members are referenced if _warn is True

class rdflib.namespace.**Namespace**(*value: Union[str, bytes]*)

Bases: str

Utility class for quickly generating URIRefs with a common prefix

```
>>> from rdflib.namespace import Namespace
>>> n = Namespace("http://example.org/")
>>> n.Person # as attribute
rdflib.term.URIRef('http://example.org/Person')
>>> n['first-name'] # as item - for things that are not valid python identifiers
rdflib.term.URIRef('http://example.org/first-name')
>>> n.Person in n
True
>>> n2 = Namespace("http://example2.org/")
>>> n.Person in n2
False
```

__annotations__ = {}

__contains__(*ref*)

Allows to check if a URI is within (starts with) this Namespace.

```
>>> from rdflib import URIRef
>>> namespace = Namespace('http://example.org/')
>>> uri = URIRef('http://example.org/foo')
>>> uri in namespace
True
>>> person_class = namespace['Person']
>>> person_class in namespace
True
>>> obj = URIRef('http://not.example.org/bar')
>>> obj in namespace
False
```

Parameters
ref (str) –

Return type
bool

__dict__ = mappingproxy({'__module__': 'rdflib.namespace', '__doc__': '\n Utility class for quickly generating URIRefs with a common prefix\n\n >>> from rdflib.namespace import Namespace\n >>> n = Namespace("http://example.org/")\n >>> n.Person # as attribute\n rdflib.term.URIRef(\'http://example.org/Person\')\n >>> n[\'first-name\'] # as item - for things that are not valid python identifiers\n rdflib.term.URIRef(\'http://example.org/first-name\')\n >>> n.Person in n\n True\n >>> n2 = Namespace("http://example2.org/")\n >>> n.Person in n2\n False\n ', '__new__': <staticmethod object>, 'title': <property object>, 'term': <function Namespace.term>, '__getitem__': <function Namespace.__getitem__>, '__getattr__': <function Namespace.__getattr__>, '__repr__': <function Namespace.__repr__>, '__contains__': <function Namespace.__contains__>, '__dict__': <attribute '__dict__' of 'Namespace' objects>, '__weakref__': <attribute '__weakref__' of 'Namespace' objects>, '__annotations__': {}})

__getattr__(*name*)

Parameters
name (str) –

Return type
URIRef

__getitem__(*key*)

Return self[key].

Parameters
key (str) –

Return type
URIRef

__module__ = 'rdflib.namespace'

static __new__(*cls*, *value*)

Parameters
value (Union[str, bytes]) –

Return type
Namespace

__repr__()
Return repr(self).

Return type
str

__weakref__
list of weak references to the object (if defined)

term(*name*)

Parameters
name (str) –

Return type
URIRef

property title: *URIRef*
Return a version of the string where each word is titlecased.

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

Return type
URIRef

class rdflib.namespace.**NamespaceManager**(*graph*, *bind_namespaces='core'*)
Bases: object

Class for managing prefix => namespace mappings

This class requires an RDFlib Graph as an input parameter and may optionally have the parameter bind_namespaces set. This second parameter selects a strategy which is one of the following:

- **core:**
  - binds several core RDF prefixes only
  - owl, rdf, rdfs, xsd, xml from the NAMESPACE_PREFIXES_CORE object
  - this is default
- **rdflib:**
  - binds all the namespaces shipped with RDFLib as DefinedNamespace instances
  - all the core namespaces and all the following: brick, csvw, dc, dcat
  - dcmitype, dcterms, dcam, doap, foaf, geo, odrl, org, prof, prov, qb, sdo
  - sh, skos, sosa, ssn, time, vann, void
  - see the NAMESPACE_PREFIXES_RDFLIB object for the up-to-date list
- **none:**
  - binds no namespaces to prefixes
  - note this is NOT default behaviour
- **cc:**
  - using prefix bindings from prefix.cc which is a online prefixes database

- not implemented yet - this is aspirational

See the Sample usage

```
>>> import rdflib
>>> from rdflib import Graph
>>> from rdflib.namespace import Namespace, NamespaceManager
>>> EX = Namespace('http://example.com/')
>>> namespace_manager = NamespaceManager(Graph())
>>> namespace_manager.bind('ex', EX, override=False)
>>> g = Graph()
>>> g.namespace_manager = namespace_manager
>>> all_ns = [n for n in g.namespace_manager.namespaces()]
>>> assert ('ex', rdflib.term.URIRef('http://example.com/')) in all_ns
>>>
```

Parameters

- **graph** (*Graph*) –
- **bind_namespaces** (Literal['core', 'rdflib', 'none']) –

__contains__(*ref*)

Parameters
ref (str) –

Return type
bool

__dict__ = mappingproxy({'__module__': 'rdflib.namespace', '__doc__': "Class for managing prefix => namespace mappings\n\n This class requires an RDFlib Graph as an input parameter and may optionally have\n the parameter bind_namespaces set. This second parameter selects a strategy which\n is one of the following:\n\n * core:\n * binds several core RDF prefixes only\n * owl, rdf, rdfs, xsd, xml from the NAMESPACE_PREFIXES_CORE object\n * this is default\n * rdflib:\n * binds all the namespaces shipped with RDFLib as DefinedNamespace instances\n * all the core namespaces and all the following: brick, csvw, dc, dcat\n * dcmitype, dcterms, dcam, doap, foaf, geo, odrl, org, prof, prov, qb, sdo\n * sh, skos, sosa, ssn, time, vann, void\n * see the NAMESPACE_PREFIXES_RDFLIB object for the up-to-date list\n * none:\n * binds no namespaces to prefixes\n * note this is NOT default behaviour\n * cc:\n * using prefix bindings from prefix.cc which is a online prefixes database\n * not implemented yet - this is aspirational\n\n See the\n Sample usage\n\n .. code-block:: pycon\n\n >>> import rdflib\n >>> from rdflib import Graph\n >>> from rdflib.namespace import Namespace, NamespaceManager\n >>> EX = Namespace('http://example.com/')\n >>> namespace_manager = NamespaceManager(Graph())\n >>> namespace_manager.bind('ex', EX, override=False)\n >>> g = Graph()\n >>> g.namespace_manager = namespace_manager\n >>> all_ns = [n for n in g.namespace_manager.namespaces()]\n >>> assert ('ex', rdflib.term.URIRef('http://example.com/')) in all_ns\n >>>\n\n ", '__init__': <function NamespaceManager.__init__>, '__contains__': <function NamespaceManager.__contains__>, 'reset': <function NamespaceManager.reset>, 'store': <property object>, 'qname': <function NamespaceManager.qname>, 'qname_strict': <function NamespaceManager.qname_strict>, 'normalizeUri': <function NamespaceManager.normalizeUri>, 'compute_qname': <function NamespaceManager.compute_qname>, 'compute_qname_strict': <function NamespaceManager.compute_qname_strict>, 'expand_curie': <function NamespaceManager.expand_curie>, '_store_bind': <function NamespaceManager._store_bind>, 'bind': <function NamespaceManager.bind>, 'namespaces': <function NamespaceManager.namespaces>, 'absolutize': <function NamespaceManager.absolutize>, '__dict__': <attribute '__dict__' of 'NamespaceManager' objects>, '__weakref__': <attribute '__weakref__' of 'NamespaceManager' objects>, '__annotations__': {'__cache': 'Dict[str, Tuple[str, URIRef, str]]', '__cache_strict': 'Dict[str, Tuple[str, URIRef, str]]', '__strie': 'Dict[str, Any]', '__trie': 'Dict[str, Any]'}})

__init__(*graph*, *bind_namespaces='core'*)

Parameters

- **graph** (*Graph*) –
- **bind_namespaces** (Literal['core', 'rdflib', 'none']) –

__module__ = 'rdflib.namespace'

__weakref__

list of weak references to the object (if defined)

absolutize(*uri*, *defrag=1*)

Parameters

- **uri** (str) –
- **defrag** (int) –

Return type
URIRef

bind(*prefix*, *namespace*, *override=True*, *replace=False*)

Bind a given namespace to the prefix

If override, rebind, even if the given namespace is already bound to another prefix.

If replace, replace any existing prefix with the new namespace

Parameters

- **prefix** (Optional[str]) –
- **namespace** (Any) –
- **override** (bool) –
- **replace** (bool) –

Return type
None

compute_qname(*uri*, *generate=True*)

Parameters

- **uri** (str) –
- **generate** (bool) –

Return type
Tuple[str, *URIRef*, str]

compute_qname_strict(*uri*, *generate=True*)

Parameters

- **uri** (str) –
- **generate** (bool) –

Return type
Tuple[str, str, str]

expand_curie(*curie*)

Expand a CURIE of the form <prefix:element>, e.g. “rdf:type” into its full expression:

```
>>> import rdflib
>>> g = rdflib.Graph()
>>> g.namespace_manager.expand_curie("rdf:type")
rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type')
```

Raises exception if a namespace is not bound to the prefix.

Parameters
curie (str) –

Return type
URIRef

namespaces()

Return type
Iterable[Tuple[str, *URIRef*]]

normalizeUri(*rdfTerm*)

Takes an RDF Term and 'normalizes' it into a QName (using the registered prefix) or (unlike compute_qname) the Notation 3 form for URIs: <...URI...>

Parameters
rdfTerm (str) –

Return type
str

qname(*uri*)

Parameters
uri (str) –

Return type
str

qname_strict(*uri*)

Parameters
uri (str) –

Return type
str

reset()

Return type
None

property store: *Store*

Return type
Store

rdflib.namespace.**is_ncname**(*name*)

Parameters
name (str) –

Return type
int

rdflib.namespace.**split_uri**(*uri*, *split_start=['Ll', 'Lu', 'Lo', 'Lt', 'Nl', 'Nd']*)

Parameters

- **uri** (str) –
- **split_start** (List[str]) –

Return type
Tuple[str, str]

rdflib.plugins package

Subpackages

rdflib.plugins.parsers package

Submodules

rdflib.plugins.parsers.RDFVOC module

class rdflib.plugins.parsers.RDFVOC.**RDFVOC**

Bases: *RDF*

Description: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#Description')

ID: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#ID')

RDF: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#RDF')

about: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#about')

datatype: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#datatype')

li: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#li')

nodeID: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#nodeID')

parseType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#parseType')

resource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#resource')

rdflib.plugins.parsers.hext module

This is a rdflib plugin for parsing Hextuple files, which are Newline-Delimited JSON (ndjson) files, into Conjunctive. The store that backs the graph *must* be able to handle contexts, i.e. multiple graphs.

class rdflib.plugins.parsers.hext.**HextuplesParser**

Bases: *Parser*

An RDFLib parser for Hextuples

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.hext', '__doc__': '\n An RDFLib parser for Hextuples\n\n ', '__init__': <function HextuplesParser.__init__>, '_load_json_line': <function HextuplesParser._load_json_line>, '_parse_hextuple': <function HextuplesParser._parse_hextuple>, 'parse': <function HextuplesParser.parse>, '__dict__': <attribute '__dict__' of 'HextuplesParser' objects>, '__weakref__': <attribute '__weakref__' of 'HextuplesParser' objects>, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.hext'

__weakref__
list of weak references to the object (if defined)

parse(*source*, *graph*, ***kwargs*)

Parameters

- **source** (*InputSource*) –
- **graph** (*Graph*) –
- **kwargs** (*Any*) –

Return type
None

rdflib.plugins.parsers.jsonld module

This parser will interpret a JSON-LD document as an RDF Graph. See:

http://json-ld.org/

Example usage:

```
>>> from rdflib import Graph, URIRef, Literal
>>> test_json = '''
... {
...     "@context": {
...         "dc": "http://purl.org/dc/terms/",
...         "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
...         "rdfs": "http://www.w3.org/2000/01/rdf-schema#"
...     },
...     "@id": "http://example.org/about",
...     "dc:title": {
...         "@language": "en",
...         "@value": "Someone's Homepage"
...     }
... }
... '''
>>> g = Graph().parse(data=test_json, format='json-ld')
>>> list(g) == [(URIRef('http://example.org/about'),
...     URIRef('http://purl.org/dc/terms/title'),
...     Literal("Someone's Homepage", lang='en'))]
True
```

class rdflib.plugins.parsers.jsonld.JsonLDParser

Bases: *Parser*

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.jsonld', '__init__': <function JsonLDParser.__init__>, 'parse': <function JsonLDParser.parse>, '__dict__': <attribute '__dict__' of 'JsonLDParser' objects>, '__weakref__': <attribute '__weakref__' of 'JsonLDParser' objects>, '__doc__': None, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.jsonld'

__weakref__
list of weak references to the object (if defined)

parse(*source*, *sink*, ***kwargs*)

Parameters

- **source** (*InputSource*) –
- **sink** (*Graph*) –
- **kwargs** (Any) –

Return type
None

rdflib.plugins.parsers.jsonld.**to_rdf**(*data*, *dataset*, *base=None*, *context_data=None*, *version=None*, *generalized_rdf=False*, *allow_lists_of_lists=None*)

Parameters

- **data** (Any) –
- **dataset** (*Graph*) –
- **base** (Optional[str]) –
- **context_data** (Optional[bool]) –
- **version** (Optional[float]) –
- **generalized_rdf** (bool) –
- **allow_lists_of_lists** (Optional[bool]) –

rdflib.plugins.parsers.notation3 module

notation3.py - Standalone Notation3 Parser Derived from CWM, the Closed World Machine

Authors of the original suite:

- Dan Connolly <@@>
- Tim Berners-Lee <@@>
- Yosi Scharf <@@>
- Joseph M. Reagle Jr. <reagle@w3.org>
- Rich Salz <rsalz@zolera.com>

http://www.w3.org/2000/10/swap/notation3.py

Copyright 2000-2007, World Wide Web Consortium. Copyright 2001, MIT. Copyright 2001, Zolera Systems Inc.

License: W3C Software License http://www.w3.org/Consortium/Legal/copyright-software

Modified by Sean B. Palmer Copyright 2007, Sean B. Palmer.

Modified to work with rdflib by Gunnar Aastrand Grimnes Copyright 2010, Gunnar A. Grimnes

exception rdflib.plugins.parsers.notation3.**BadSyntax**(*uri*, *lines*, *argstr*, *i*, *why*)

Bases: SyntaxError

Parameters

- **uri** (str) –
- **lines** (int) –
- **argstr** (str) –
- **i** (int) –
- **why** (str) –

__init__(*uri*, *lines*, *argstr*, *i*, *why*)

Parameters

- **uri** (str) –
- **lines** (int) –
- **argstr** (str) –
- **i** (int) –
- **why** (str) –

__module__ = 'rdflib.plugins.parsers.notation3'

__str__()

Return str(self).

Return type
str

__weakref__

list of weak references to the object (if defined)

property message: str

Return type
str

class rdflib.plugins.parsers.notation3.**Formula**(*parent*)

Bases: object

Parameters
parent (*Graph*) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.notation3', 'number': 0, '__init__': <function Formula.__init__>, '__str__': <function Formula.__str__>, 'id': <function Formula.id>, 'newBlankNode': <function Formula.newBlankNode>, 'newUniversal': <function Formula.newUniversal>, 'declareExistential': <function Formula.declareExistential>, 'close': <function Formula.close>, '__dict__': <attribute '__dict__' of 'Formula' objects>, '__weakref__': <attribute '__weakref__' of 'Formula' objects>, '__doc__': None, '__annotations__': {'existentials': 'Dict[str, BNode]', 'universals': 'Dict[str, BNode]'}})

__init__(*parent*)

Parameters
parent (*Graph*) –

__module__ = 'rdflib.plugins.parsers.notation3'

__str__()
Return str(self).

Return type
str

__weakref__
list of weak references to the object (if defined)

close()

Return type
QuotedGraph

declareExistential(*x*)

Parameters
x (str) –

Return type
None

id()

Return type
BNode

newBlankNode(*uri=None*, *why=None*)

Parameters
- **uri** (Optional[str]) –
- **why** (Optional[Any]) –

Return type
BNode

newUniversal(*uri*, *why=None*)

Parameters
- **uri** (str) –
- **why** (Optional[Any]) –

Return type
Variable

number = 0

class rdflib.plugins.parsers.notation3.**N3Parser**
Bases: *TurtleParser*

An RDFLib parser for Notation3

See http://www.w3.org/DesignIssues/Notation3.html

__init__()

__module__ = 'rdflib.plugins.parsers.notation3'

parse(*source*, *graph*, *encoding='utf-8'*)

Parameters

- **source** (*InputSource*) –
- **graph** (*Graph*) –
- **encoding** (Optional[str]) –

Return type
None

class rdflib.plugins.parsers.notation3.**RDFSink**(*graph*)

Bases: object

Parameters
graph (*Graph*) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.notation3', '__init__': <function RDFSink.__init__>, 'newFormula': <function RDFSink.newFormula>, 'newGraph': <function RDFSink.newGraph>, 'newSymbol': <function RDFSink.newSymbol>, 'newBlankNode': <function RDFSink.newBlankNode>, 'newLiteral': <function RDFSink.newLiteral>, 'newList': <function RDFSink.newList>, 'newSet': <function RDFSink.newSet>, 'setDefaultNamespace': <function RDFSink.setDefaultNamespace>, 'makeStatement': <function RDFSink.makeStatement>, 'normalise': <function RDFSink.normalise>, 'intern': <function RDFSink.intern>, 'bind': <function RDFSink.bind>, 'startDoc': <function RDFSink.startDoc>, 'endDoc': <function RDFSink.endDoc>, '__dict__': <attribute '__dict__' of 'RDFSink' objects>, '__weakref__': <attribute '__weakref__' of 'RDFSink' objects>, '__doc__': None, '__annotations__': {'rootFormula': 'Optional[Formula]'}})

__init__(*graph*)

Parameters
graph (*Graph*) –

__module__ = 'rdflib.plugins.parsers.notation3'

__weakref__
list of weak references to the object (if defined)

bind(*pfx*, *uri*)

Return type
None

endDoc(*formula*)

Parameters
formula (Optional[*Formula*]) –

Return type
None

intern(*something*)

Parameters
something (TypeVar(_AnyT)) –

Return type
TypeVar(_AnyT)

makeStatement(*quadruple*, *why=None*)

Parameters

- **quadruple** (Tuple[Union[*Formula*, *Graph*, None], *Node*, *Node*, *Node*]) –
- **why** (Optional[Any]) –

Return type
None

newBlankNode(*arg=None*, *uri=None*, *why=None*)

Parameters

- **arg** (Union[*Formula*, *Graph*, Any, None]) –
- **uri** (Optional[str]) –
- **why** (Optional[Callable[[], None]]) –

Return type
BNode

newFormula()

Return type
Formula

newGraph(*identifier*)

Parameters
identifier (*Identifier*) –

Return type
Graph

newList(*n*, *f*)

Parameters

- **n** (List[Any]) –
- **f** (Optional[*Formula*]) –

Return type
IdentifiedNode

newLiteral(*s*, *dt*, *lang*)

Parameters

- **s** (str) –
- **dt** (Optional[*URIRef*]) –
- **lang** (Optional[str]) –

Return type
Literal

newSet(**args*)

Parameters
args (TypeVar(_AnyT)) –

Return type
Set[TypeVar(_AnyT)]

newSymbol(**args*)

Parameters
args (str) –

Return type
URIRef

normalise(*f*, *n*)

Parameters

- **f** (Optional[*Formula*]) –
- **n** (Union[Tuple[int, str], bool, int, Decimal, float, TypeVar(_AnyT)]) –

Return type
Union[*URIRef*, *Literal*, *BNode*, TypeVar(_AnyT)]

setDefaultNamespace(**args*)

Parameters
args (bytes) –

Return type
str

startDoc(*formula*)

Parameters
formula (Optional[*Formula*]) –

Return type
None

class rdflib.plugins.parsers.notation3.**SinkParser**(*store*, *openFormula=None*, *thisDoc=''*, *baseURI=None*, *genPrefix=''*, *why=None*, *turtle=False*)

Bases: *object*

Parameters

- **store** (*RDFSink*) –
- **openFormula** (Optional[*Formula*]) –
- **thisDoc** (str) –
- **baseURI** (Optional[str]) –
- **genPrefix** (str) –
- **why** (Optional[Callable[[], None]]) –

- **turtle** (bool) –

BadSyntax(*argstr*, *i*, *msg*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **msg** (str) –

Return type
NoReturn

UEscape(*argstr*, *i*, *startline*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **startline** (int) –

Return type
Tuple[int, str]

```
__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.notation3',
'__init__': <function SinkParser.__init__>, 'here': <function SinkParser.here>,
'formula': <function SinkParser.formula>, 'loadStream': <function
SinkParser.loadStream>, 'loadBuf': <function SinkParser.loadBuf>, 'feed':
<function SinkParser.feed>, 'directiveOrStatement': <function
SinkParser.directiveOrStatement>, 'tok': <function SinkParser.tok>, 'sparqlTok':
<function SinkParser.sparqlTok>, 'directive': <function SinkParser.directive>,
'sparqlDirective': <function SinkParser.sparqlDirective>, 'bind': <function
SinkParser.bind>, 'setKeywords': <function SinkParser.setKeywords>, 'startDoc':
<function SinkParser.startDoc>, 'endDoc': <function SinkParser.endDoc>,
'makeStatement': <function SinkParser.makeStatement>, 'statement': <function
SinkParser.statement>, 'subject': <function SinkParser.subject>, 'verb': <function
SinkParser.verb>, 'prop': <function SinkParser.prop>, 'item': <function
SinkParser.item>, 'blankNode': <function SinkParser.blankNode>, 'path': <function
SinkParser.path>, 'anonymousNode': <function SinkParser.anonymousNode>, 'node':
<function SinkParser.node>, 'property_list': <function SinkParser.property_list>,
'commaSeparatedList': <function SinkParser.commaSeparatedList>, 'objectList':
<function SinkParser.objectList>, 'checkDot': <function SinkParser.checkDot>,
'uri_ref2': <function SinkParser.uri_ref2>, 'skipSpace': <function
SinkParser.skipSpace>, 'variable': <function SinkParser.variable>, 'bareWord':
<function SinkParser.bareWord>, 'qname': <function SinkParser.qname>, 'object':
<function SinkParser.object>, 'nodeOrLiteral': <function SinkParser.nodeOrLiteral>,
'uriOf': <function SinkParser.uriOf>, 'strconst': <function SinkParser.strconst>,
'_unicodeEscape': <function SinkParser._unicodeEscape>, 'uEscape': <function
SinkParser.uEscape>, 'UEscape': <function SinkParser.UEscape>, 'BadSyntax':
<function SinkParser.BadSyntax>, '__dict__': <attribute '__dict__' of 'SinkParser'
objects>, '__weakref__': <attribute '__weakref__' of 'SinkParser' objects>,
'__doc__': None, '__annotations__': {'_anonymousNodes': 'Dict[str, BNode]',
'_variables': 'Dict[str, Variable]', '_parentVariables': 'Dict[str, Variable]',
'_reason2': 'Optional[Callable[..., None]]', '_baseURI': 'Optional[str]',
'_formula': 'Optional[Formula]', '_context': 'Optional[Formula]',
'_parentContext': 'Optional[Formula]'}})
```

__init__(*store*, *openFormula=None*, *thisDoc=''*, *baseURI=None*, *genPrefix=''*, *why=None*, *turtle=False*)

note: namespace names should *not* end in # ; the # will get added during qname processing

Parameters

- **store** (*RDFSink*) –
- **openFormula** (Optional[*Formula*]) –
- **thisDoc** (str) –
- **baseURI** (Optional[str]) –
- **genPrefix** (str) –
- **why** (Optional[Callable[[], None]]) –
- **turtle** (bool) –

__module__ = 'rdflib.plugins.parsers.notation3'

__weakref__

list of weak references to the object (if defined)

anonymousNode(*ln*)

Remember or generate a term for one of these _: anonymous nodes

Parameters
ln (str) –

Return type
BNode

bareWord(*argstr*, *i*, *res*)

abc -> :abc

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

bind(*qn*, *uri*)

Parameters

- **qn** (str) –
- **uri** (bytes) –

Return type
None

blankNode(*uri=None*)

Parameters
uri (Optional[str]) –

Return type
BNode

checkDot(*argstr*, *i*)

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

commaSeparatedList(*argstr*, *j*, *res*, *what*)

return value: -1 bad syntax; >1 new position in argstr res has things found appended

Parameters

- **argstr** (str) –
- **j** (int) –
- **res** (MutableSequence[Any]) –
- **what** (Callable[[str, int, MutableSequence[Any]], int]) –

Return type
int

directive(*argstr*, *i*)

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

directiveOrStatement(*argstr*, *h*)

Parameters

- **argstr** (str) –
- **h** (int) –

Return type
int

endDoc()

Signal end of document and stop parsing. returns formula

Return type
Optional[*Formula*]

feed(*octets*)

Feed an octet stream to the parser

if BadSyntax is raised, the string passed in the exception object is the remainder after any statements have been parsed. So if there is more data to feed to the parser, it should be straightforward to recover.

Parameters
octets (Union[str, bytes]) –

Return type
None

formula()

Return type
Optional[*Formula*]

here(*i*)

String generated from position in file

This is for repeatability when referring people to bnodes in a document. This has diagnostic uses less formally, as it should point one to which bnode the arbitrary identifier actually is. It gives the line and character number of the '[' charcacter or path character which introduced the blank node. The first blank node is boringly _L1C1. It used to be used only for tracking, but for tests in general it makes the canonical ordering of bnodes repeatable.

Parameters
i (int) –

Return type
str

item(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **res** (MutableSequence[Any]) –

Return type
int

loadBuf(*buf*)

Parses a buffer and returns its top level formula

Parameters
buf (Union[str, bytes]) –

Return type
Optional[*Formula*]

loadStream(*stream*)

Parameters
stream (Union[IO[str], IO[bytes]]) –

Return type
Optional[*Formula*]

makeStatement(*quadruple*)

Return type
None

node(*argstr*, *i*, *res*, *subjectAlready=None*)

Parse the <node> production. Space is now skipped once at the beginning instead of in multiple calls to self.skipSpace().

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

- **subjectAlready** (Optional[*Node*]) –

Return type
int

nodeOrLiteral(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

object(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

objectList(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

path(*argstr*, *i*, *res*)

Parse the path production.

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

prop(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

property_list(*argstr*, *i*, *subj*)

Parse property list Leaves the terminating punctuation in the buffer

Parameters

- **argstr** (str) –
- **i** (int) –
- **subj** (*Node*) –

Return type
int

qname(*argstr*, *i*, *res*)

xyz:def -> (‘xyz’, ‘def’) If not in keywords and keywordsSet: def -> (‘’, ‘def’) :def -> (‘’, ‘def’)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Union[*Identifier*, Tuple[str, str]]]) –

Return type
int

setKeywords(*k*)

Takes a list of strings

Parameters
k (Optional[List[str]]) –

Return type
None

skipSpace(*argstr*, *i*)

Skip white space, newlines and comments. return -1 if EOF, else position of first non-ws character

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

sparqlDirective(*argstr*, *i*)

turtle and trig support BASE/PREFIX without @ and without terminating .

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

sparqlTok(*tok*, *argstr*, *i*)

Check for SPARQL keyword. Space must have been stripped on entry and we must not be at end of file. Case insensitive and not preceded by @

Parameters

- **tok** (str) –
- **argstr** (str) –
- **i** (int) –

Return type
int

startDoc()

Return type
None

statement(*argstr*, *i*)

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

strconst(*argstr*, *i*, *delim*)

parse an N3 string constant delimited by delim. return index, val

Parameters

- **argstr** (str) –
- **i** (int) –
- **delim** (str) –

Return type
Tuple[int, str]

subject(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

tok(*tok*, *argstr*, *i*, *colon=False*)

Check for keyword. Space must have been stripped on entry and we must not be at end of file.

if colon, then keyword followed by colon is ok (@prefix:<blah> is ok, rdf:type shortcut a must be followed by ws)

Parameters

- **tok** (str) –

- **argstr** (str) –
- **i** (int) –
- **colon** (bool) –

Return type
int

uEscape(*argstr*, *i*, *startline*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **startline** (int) –

Return type
Tuple[int, str]

uriOf(*sym*)

Parameters
sym (Union[*Identifier*, Tuple[str, str]]) –

Return type
str

uri_ref2(*argstr*, *i*, *res*)

Generate uri from n3 representation.

Note that the RDF convention of directly concatenating NS and local name is now used though I prefer inserting a '#' to make the namesapces look more like what XML folks expect.

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

variable(*argstr*, *i*, *res*)

?abc -> variable(:abc)

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

verb(*argstr*, *i*, *res*)

has _prop_ is _prop_ of a = _prop_ >- prop -> <- prop -< _operator_

Parameters

- **argstr** (str) –
- **i** (int) –

- **res** (MutableSequence[Any]) –

Return type
int

class rdflib.plugins.parsers.notation3.**TurtleParser**

Bases: *Parser*

An RDFLib parser for Turtle

See http://www.w3.org/TR/turtle/

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.notation3', '__doc__': '\n An RDFLib parser for Turtle\n\n See http://www.w3.org/TR/turtle/\n ', '__init__': <function TurtleParser.__init__>, 'parse': <function TurtleParser.parse>, '__dict__': <attribute '__dict__' of 'TurtleParser' objects>, '__weakref__': <attribute '__weakref__' of 'TurtleParser' objects>, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.notation3'

__weakref__

list of weak references to the object (if defined)

parse(*source*, *graph*, *encoding='utf-8'*, *turtle=True*)

Parameters

- **source** (*InputSource*) –
- **graph** (*Graph*) –
- **encoding** (Optional[str]) –
- **turtle** (bool) –

Return type
None

rdflib.plugins.parsers.notation3.**base**()

The base URI for this process - the Web equiv of cwd

Relative or absolute unix-standard filenames parsed relative to this yield the URI of the file. If we had a reliable way of getting a computer name, we should put it in the hostname just to prevent ambiguity

Return type
str

rdflib.plugins.parsers.notation3.**hexify**(*ustr*)

Use URL encoding to return an ASCII string corresponding to the given UTF8 string

```
>>> hexify("http://example/a b")
b'http://example/a%20b'
```

Parameters
ustr (str) –

Return type
bytes

rdflib.plugins.parsers.notation3.**join**(*here*, *there*)

join an absolute URI and URI reference (non-ascii characters are supported/doctested; haven't checked the details of the IRI spec though)

`here` is assumed to be absolute. `there` is URI reference.

```
>>> join('http://example/x/y/z', '../abc')
'http://example/x/abc'
```

Raise ValueError if there uses relative path syntax but here has no hierarchical path.

```
>>> join('mid:foo@example', '../foo')
Traceback (most recent call last):
    raise ValueError(here)
ValueError: Base <mid:foo@example> has no slash
after colon - with relative '../foo'.
```

```
>>> join('http://example/x/y/z', '')
'http://example/x/y/z'
```

```
>>> join('mid:foo@example', '#foo')
'mid:foo@example#foo'
```

We grok IRIs

```
>>> len(u'Andr\xe9')
5
```

```
>>> join('http://example.org/', u'#Andr\xe9')
u'http://example.org/#Andr\xe9'
```

Parameters

- **here** (str) –
- **there** (str) –

Return type

str

rdflib.plugins.parsers.notation3.**runNamespace**()

Returns a URI suitable as a namespace for run-local objects

Return type

str

rdflib.plugins.parsers.notation3.**splitFragP**(*uriref*, *punc=0*)

split a URI reference before the fragment

Punctuation is kept.

e.g.

```
>>> splitFragP("abc#def")
('abc', '#def')
```

```
>>> splitFragP("abcdef")
('abcdef', '')
```

Parameters

- **uriref** (str) –
- **punc** (int) –

Return type
Tuple[str, str]

rdflib.plugins.parsers.notation3.**uniqueURI**()

A unique URI

Return type
str

rdflib.plugins.parsers.nquads module

This is a rdflib plugin for parsing NQuad files into Conjunctive graphs that can be used and queried. The store that backs the graph *must* be able to handle contexts.

```
>>> from rdflib import ConjunctiveGraph, URIRef, Namespace
>>> g = ConjunctiveGraph()
>>> data = open("test/data/nquads.rdflib/example.nquads", "rb")
>>> g.parse(data, format="nquads")
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> assert len(g.store) == 449
>>> # There should be 16 separate contexts
>>> assert len([x for x in g.store.contexts()]) == 16
>>> # is the name of entity E10009 "Arco Publications"?
>>> #   (in graph http://bibliographica.org/entity/E10009)
>>> # Looking for:
>>> # <http://bibliographica.org/entity/E10009>
>>> #   <http://xmlns.com/foaf/0.1/name>
>>> #   "Arco Publications"
>>> #   <http://bibliographica.org/entity/E10009>
>>> s = URIRef("http://bibliographica.org/entity/E10009")
>>> FOAF = Namespace("http://xmlns.com/foaf/0.1/")
>>> assert(g.value(s, FOAF.name).eq("Arco Publications"))
```

class rdflib.plugins.parsers.nquads.**NQuadsParser**(*sink=None*, *bnode_context=None*)

Bases: *W3CNTriplesParser*

Parameters

- **sink** (Union[*DummySink*, *NTGraphSink*, None]) –
- **bnode_context** (Optional[MutableMapping[str, *BNode*]]) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.nquads', 'parse': <function NQuadsParser.parse>, 'parseline': <function NQuadsParser.parseline>, '__dict__': <attribute '__dict__' of 'NQuadsParser' objects>, '__weakref__': <attribute '__weakref__' of 'NQuadsParser' objects>, '__doc__': None, '__annotations__': {'sink': "Union[DummySink, 'NTGraphSink']", 'buffer': 'Optional[str]', 'file': 'Optional[Union[TextIO, codecs.StreamReader]]', 'line': 'Optional[str]'}})

__module__ = 'rdflib.plugins.parsers.nquads'

__weakref__

list of weak references to the object (if defined)

buffer: Optional[str]

file: Optional[Union[TextIO, codecs.StreamReader]]

line: Optional[str]

parse(*inputsource*, *sink*, *bnode_context=None*, ***kwargs*)

Parse inputsource as an N-Quads file.

Parameters

- **inputsource** (*InputSource*) – the source of N-Quads-formatted data
- **sink** (*ConjunctiveGraph*) – where to send parsed triples
- **bnode_context** (Optional[MutableMapping[str, *BNode*]]) – a dict mapping blank node identifiers to *BNode* instances. See *W3CNTriplesParser.parse*
- **kwargs** (Any) –

Return type

ConjunctiveGraph

parseline(*bnode_context=None*)

Parameters

bnode_context (Optional[MutableMapping[str, *BNode*]]) –

Return type

None

sink: Union[*DummySink*, 'NTGraphSink']

rdflib.plugins.parsers.ntriples module

N-Triples Parser License: GPL 2, W3C, BSD, or MIT Author: Sean B. Palmer, inamidst.com

class rdflib.plugins.parsers.ntriples.**DummySink**

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.ntriples', '__init__': <function DummySink.__init__>, 'triple': <function DummySink.triple>, '__dict__': <attribute '__dict__' of 'DummySink' objects>, '__weakref__': <attribute '__weakref__' of 'DummySink' objects>, '__doc__': None, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.ntriples'

__weakref__

list of weak references to the object (if defined)

triple(*s*, *p*, *o*)

class rdflib.plugins.parsers.ntriples.NTGraphSink(*graph*)

Bases: object

Parameters
graph (*Graph*) –

__init__(*graph*)

Parameters
graph (*Graph*) –

__module__ = 'rdflib.plugins.parsers.ntriples'

__slots__ = ('g',)

g

triple(*s*, *p*, *o*)

Parameters

- **s** (*Node*) –
- **p** (*Node*) –
- **o** (*Node*) –

Return type
None

class rdflib.plugins.parsers.ntriples.NTParser

Bases: *Parser*

parser for the ntriples format, often stored with the .nt extension

See http://www.w3.org/TR/rdf-testcases/#ntriples

__module__ = 'rdflib.plugins.parsers.ntriples'

__slots__ = ()

classmethod parse(*source*, *sink*, ***kwargs*)

Parse the NT format

Parameters

- **source** (*InputSource*) – the source of NT-formatted data
- **sink** (*Graph*) – where to send parsed triples
- **kwargs** (*Any*) – Additional arguments to pass to *W3CNTriplesParser.parse*

Return type
None

class rdflib.plugins.parsers.ntriples.**W3CNTriplesParser**(*sink=None*, *bnode_context=None*)

Bases: *object*

An N-Triples Parser. This is a legacy-style Triples parser for NTriples provided by W3C Usage:

```
p = W3CNTriplesParser(sink=MySink())
sink = p.parse(f) # file; use parsestring for a string
```

To define a context in which blank node identifiers refer to the same blank node across instances of NTriplesParser, pass the same dict as `bnode_context` to each instance. By default, a new blank node context is created for each instance of *W3CNTriplesParser*.

Parameters

- **sink** (Union[*DummySink*, *NTGraphSink*, None]) –
- **bnode_context** (Optional[MutableMapping[str, *BNode*]]) –

__init__(*sink=None*, *bnode_context=None*)

Parameters

- **sink** (Union[*DummySink*, *NTGraphSink*, None]) –
- **bnode_context** (Optional[MutableMapping[str, *BNode*]]) –

__module__ = 'rdflib.plugins.parsers.ntriples'

__slots__ = ('_bnode_ids', 'sink', 'buffer', 'file', 'line')

buffer: Optional[str]

eat(*pattern*)

Parameters
pattern (Pattern[str]) –

Return type
Match[str]

file: Optional[Union[TextIO, StreamReader]]

line: Optional[str]

literal()

Return type
Union[Literal[False], *Literal*]

nodeid(*bnode_context=None*)

Parameters
bnode_context (Optional[MutableMapping[str, *BNode*]]) –

Return type
Union[Literal[False], *BNode*]

object(*bnode_context=None*)

Parameters
bnode_context (Optional[MutableMapping[str, *BNode*]]) –

Return type
Union[*URIRef*, *BNode*, *Literal*]

parse(*f*, *bnode_context=None*)

Parse f as an N-Triples file.

Parameters

- **f** (Union[TextIO, IO[bytes], StreamReader]) – the N-Triples source
- **bnode_context** (Optional[MutableMapping[str, *BNode*]]) – a dict mapping blank node identifiers (e.g., a in _:a) to *BNode* instances. An empty dict can be passed in to define a distinct context for a given call to *parse*.

Return type
Union[*DummySink*, *NTGraphSink*]

parseline(*bnode_context=None*)

Parameters
bnode_context (Optional[MutableMapping[str, *BNode*]]) –

Return type
None

parsestring(*s*, ***kwargs*)

Parse s as an N-Triples string.

Parameters
s (Union[bytes, bytearray, str]) –

Return type
None

peek(*token*)

Parameters
token (str) –

Return type
bool

predicate()

Return type
URIRef

readline()

Read an N-Triples line from buffered input.

Return type
Optional[str]

sink: Union[*DummySink*, *NTGraphSink*]

subject(*bnode_context=None*)

Return type
Union[*BNode*, *URIRef*]

uriref()

Return type
Union[Literal[False], *URIRef*]

rdflib.plugins.parsers.ntriples.**unquote**(*s*)

Unquote an N-Triples string.

Parameters
s (str) –

Return type
str

rdflib.plugins.parsers.ntriples.**uriquote**(*uri*)

Parameters
uri (str) –

Return type
str

rdflib.plugins.parsers.rdfxml module

An RDF/XML parser for RDFLib

class rdflib.plugins.parsers.rdfxml.**BagID**(*value: str*, *base: Optional[str] = None*)

Bases: *URIRef*

__init__(*val*)

__module__ = 'rdflib.plugins.parsers.rdfxml'

__slots__ = ['li']

li

next_li()

class rdflib.plugins.parsers.rdfxml.**ElementHandler**

Bases: *object*

__init__()

__module__ = 'rdflib.plugins.parsers.rdfxml'

__slots__ = ['start', 'char', 'end', 'li', 'id', 'base', 'subject', 'predicate', 'object', 'list', 'language', 'datatype', 'declared', 'data']

base

char

data

datatype

declared

end

id

language

li

list

next_li()

object

predicate

start

subject

class rdflib.plugins.parsers.rdfxml.**RDFXMLHandler**(*store*)

Bases: ContentHandler

Parameters
store (*Graph*) –

__init__(*store*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.parsers.rdfxml'

absolutize(*uri*)

Parameters
uri (str) –

Return type
URIRef

add_reified(*sid*, *spo*)

Parameters

- **sid** (*Identifier*) –
- **spo** (Tuple[*Node*, *Node*, *Node*]) –

characters(*content*)

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

Parameters
content (str) –

Return type
None

convert(*name*, *qname*, *attrs*)

Parameters

- **name** (Tuple[Optional[str], str]) –
- **attrs** (AttributesImpl) –

Return type
Tuple[*URIRef*, Dict[*URIRef*, str]]

property current: Optional[*ElementHandler*]

Return type
Optional[*ElementHandler*]

document_element_start(*name*, *qname*, *attrs*)

Parameters

- **name** (Tuple[str, str]) –
- **attrs** (AttributesImpl) –

Return type
None

endElementNS(*name*, *qname*)

Signals the end of an element in namespace mode.

The name parameter contains the name of the element type, just as with the startElementNS event.

Parameters
name (Tuple[Optional[str], str]) –

Return type
None

endPrefixMapping(*prefix*)

End the scope of a prefix-URI mapping.

See startPrefixMapping for details. This event will always occur after the corresponding endElement event, but the order of endPrefixMapping events is not otherwise guaranteed.

Parameters
prefix (Optional[str]) –

Return type
None

error(*message*)

Parameters
message (str) –

Return type
NoReturn

get_current()

Return type
Optional[*ElementHandler*]

get_next()

Return type
Optional[*ElementHandler*]

get_parent()

Return type
Optional[*ElementHandler*]

ignorableWhitespace(*content*)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

Return type
None

list_node_element_end(*name*, *qname*)

Parameters
name (Tuple[str, str]) –

Return type
None

literal_element_char(*data*)

Parameters
data (str) –

Return type
None

literal_element_end(*name*, *qname*)

Parameters
name (Tuple[str, str]) –

Return type
None

literal_element_start(*name*, *qname*, *attrs*)

Parameters

- **name** (Tuple[str, str]) –
- **attrs** (AttributesImpl) –

Return type
None

property next: Optional[*ElementHandler*]

Return type
Optional[*ElementHandler*]

node_element_end(*name*, *qname*)

Parameters
name (Tuple[str, str]) –

Return type
None

node_element_start(*name*, *qname*, *attrs*)

Parameters

- **name** (Tuple[str, str]) –
- **attrs** (AttributesImpl) –

Return type
None

property parent: Optional[*ElementHandler*]

Return type
Optional[*ElementHandler*]

processingInstruction(*target*, *data*)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

Return type
None

property_element_char(*data*)

Parameters
data (str) –

Return type
None

property_element_end(*name*, *qname*)

Parameters
name (Tuple[str, str]) –

Return type
None

property_element_start(*name*, *qname*, *attrs*)

Parameters

- **name** (Tuple[str, str]) –
- **attrs** (AttributesImpl) –

Return type
None

reset()

Return type
None

setDocumentLocator(*locator*)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

Parameters
locator (Locator) –

startDocument()

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for setDocumentLocator).

Return type
None

startElementNS(*name*, *qname*, *attrs*)

Signals the start of an element in namespace mode.

The name parameter contains the name of the element type as a (uri, localname) tuple, the qname parameter the raw XML 1.0 name used in the source document, and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

The uri part of the name tuple is None for elements which have no namespace.

Parameters

- **name** (Tuple[Optional[str], str]) –
- **attrs** (AttributesImpl) –

Return type
None

startPrefixMapping(*prefix*, *namespace*)

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the http://xml.org/sax/features/namespaces feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding startElement event, and all endPrefixMapping events will occur after the corresponding endElement event, but their order is not guaranteed.

Parameters

- **prefix** (Optional[str]) –

- **namespace** (str) –

Return type
None

class rdflib.plugins.parsers.rdfxml.**RDFXMLParser**

Bases: *Parser*

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.rdfxml', '__init__': <function RDFXMLParser.__init__>, 'parse': <function RDFXMLParser.parse>, '__dict__': <attribute '__dict__' of 'RDFXMLParser' objects>, '__weakref__': <attribute '__weakref__' of 'RDFXMLParser' objects>, '__doc__': None, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.rdfxml'

__weakref__

list of weak references to the object (if defined)

parse(*source*, *sink*, ***args*)

Parameters

- **source** (*InputSource*) –
- **sink** (*Graph*) –
- **args** (Any) –

Return type
None

rdflib.plugins.parsers.rdfxml.**create_parser**(*target*, *store*)

Parameters

- **target** (*InputSource*) –
- **store** (*Graph*) –

Return type
XMLReader

rdflib.plugins.parsers.trig module

class rdflib.plugins.parsers.trig.**TrigParser**

Bases: *Parser*

An RDFLib parser for TriG

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.trig', '__doc__': '\n An RDFLib parser for TriG\n\n ', '__init__': <function TrigParser.__init__>, 'parse': <function TrigParser.parse>, '__dict__': <attribute '__dict__' of 'TrigParser' objects>, '__weakref__': <attribute '__weakref__' of 'TrigParser' objects>, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.trig'

__weakref__
list of weak references to the object (if defined)

parse(*source*, *graph*, *encoding='utf-8'*)

Parameters

- **source** (*InputSource*) –
- **graph** (*Graph*) –
- **encoding** (str) –

Return type
None

class rdflib.plugins.parsers.trig.**TrigSinkParser**(*store*, *openFormula=None*, *thisDoc=''*, *baseURI=None*, *genPrefix=''*, *why=None*, *turtle=False*)

Bases: *SinkParser*

Parameters

- **store** (*RDFSink*) –
- **openFormula** (Optional[*Formula*]) –
- **thisDoc** (str) –
- **baseURI** (Optional[str]) –
- **genPrefix** (str) –
- **why** (Optional[Callable[[], None]]) –
- **turtle** (bool) –

__module__ = 'rdflib.plugins.parsers.trig'

directiveOrStatement(*argstr*, *h*)

Parameters

- **argstr** (str) –
- **h** (int) –

Return type
int

graph(*argstr*, *i*)

Parse trig graph, i.e.

<urn:graphname> = { .. triples .. }

return -1 if it doesn't look like a graph-decl raise Exception if it looks like a graph, but isn't.

Parameters

- **argstr** (str) –
- **i** (int) –

Return type
int

labelOrSubject(*argstr*, *i*, *res*)

Parameters

- **argstr** (str) –
- **i** (int) –
- **res** (MutableSequence[Any]) –

Return type
int

rdflib.plugins.parsers.trig.**becauseSubGraph**(**args*, ***kwargs*)

rdflib.plugins.parsers.trix module

A TriX parser for RDFLib

class rdflib.plugins.parsers.trix.**TriXHandler**(*store*)

Bases: ContentHandler

An Sax Handler for TriX. See http://sw.nokia.com/trix/

Parameters
store (*Store*) –

__init__(*store*)

Parameters
store (*Store*) –

__module__ = 'rdflib.plugins.parsers.trix'

characters(*content*)

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

Parameters
content (str) –

Return type
None

endElementNS(*name*, *qname*)

Signals the end of an element in namespace mode.

The name parameter contains the name of the element type, just as with the startElementNS event.

Parameters
name (Tuple[Optional[str], str]) –

Return type
None

endPrefixMapping(*prefix*)

End the scope of a prefix-URI mapping.

See startPrefixMapping for details. This event will always occur after the corresponding endElement event, but the order of endPrefixMapping events is not otherwise guaranteed.

Parameters
prefix (Optional[str]) –

Return type
None

error(*message*)

Parameters
message (str) –

Return type
NoReturn

get_bnode(*label*)

Parameters
label (str) –

Return type
BNode

ignorableWhitespace(*content*)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

Return type
None

processingInstruction(*target*, *data*)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

Return type
None

reset()

Return type
None

setDocumentLocator(*locator*)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

Parameters
locator (Locator) –

startDocument()

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for setDocumentLocator).

Return type
None

startElementNS(*name*, *qname*, *attrs*)

Signals the start of an element in namespace mode.

The name parameter contains the name of the element type as a (uri, localname) tuple, the qname parameter the raw XML 1.0 name used in the source document, and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

The uri part of the name tuple is None for elements which have no namespace.

Parameters

- **name** (Tuple[Optional[str], str]) –
- **attrs** (AttributesImpl) –

Return type
None

startPrefixMapping(*prefix*, *namespace*)

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the http://xml.org/sax/features/namespaces feature is true (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding startElement event, and all endPrefixMapping events will occur after the corresponding endElement event, but their order is not guaranteed.

Parameters

- **prefix** (Optional[str]) –

- **namespace** (str) –

Return type
None

class rdflib.plugins.parsers.trix.**TriXParser**

Bases: *Parser*

A parser for TriX. See http://sw.nokia.com/trix/

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.parsers.trix', '__doc__': 'A parser for TriX. See http://sw.nokia.com/trix/', '__init__': <function TriXParser.__init__>, 'parse': <function TriXParser.parse>, '__dict__': <attribute '__dict__' of 'TriXParser' objects>, '__weakref__': <attribute '__weakref__' of 'TriXParser' objects>, '__annotations__': {}})

__init__()

__module__ = 'rdflib.plugins.parsers.trix'

__weakref__

list of weak references to the object (if defined)

parse(*source*, *sink*, ***args*)

Parameters

- **source** (*InputSource*) –
- **sink** (*Graph*) –
- **args** (Any) –

Return type
None

rdflib.plugins.parsers.trix.**create_parser**(*store*)

Parameters
store (*Store*) –

Return type
XMLReader

Module contents

rdflib.plugins.serializers package

Submodules

rdflib.plugins.serializers.hext module

HextuplesSerializer RDF graph serializer for RDFLib. See <https://github.com/ontola/hextuples> for details about the format.

class rdflib.plugins.serializers.hext.**HextuplesSerializer**(*store*)

Bases: *Serializer*

Serializes RDF graphs to NTriples format.

Parameters
store (Union[*Graph*, *ConjunctiveGraph*]) –

__init__(*store*)

Parameters
store (Union[*Graph*, *ConjunctiveGraph*]) –

__module__ = 'rdflib.plugins.serializers.hext'

base: Optional[str]

encoding: str

serialize(*stream*, *base=None*, *encoding='utf-8'*, ***kwargs*)
Abstract method

Parameters
- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

store: *Graph*

rdflib.plugins.serializers.jsonld module

This serialiser will output an RDF Graph as a JSON-LD formatted document. See:

http://json-ld.org/

Example usage:

```
>>> from rdflib import Graph
>>> testrdf = '''
... @prefix dc: <http://purl.org/dc/terms/> .
... <http://example.org/about>
...     dc:title "Someone's Homepage"@en .
... '''

>>> g = Graph().parse(data=testrdf, format='n3')

>>> print(g.serialize(format='json-ld', indent=4))
[
    {
        "@id": "http://example.org/about",
        "http://purl.org/dc/terms/title": [
            {
                "@language": "en",
                "@value": "Someone's Homepage"
            }
        ]
    }
]
```

class rdflib.plugins.serializers.jsonld.**JsonLDSerializer**(*store*)

Bases: *Serializer*

Parameters
store (*Graph*) –

__init__(*store*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.jsonld'

base: Optional[str]

encoding: str

serialize(*stream*, *base=None*, *encoding=None*, ***kwargs*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

store: *Graph*

rdflib.plugins.serializers.jsonld.**from_rdf**(*graph*, *context_data=None*, *base=None*, *use_native_types=False*, *use_rdf_type=False*, *auto_compact=False*, *startnode=None*, *index=False*)

rdflib.plugins.serializers.longturtle module

LongTurtle RDF graph serializer for RDFLib. See <http://www.w3.org/TeamSubmission/turtle/> for syntax specification.

This variant, longturtle as opposed to just turtle, makes some small format changes to turtle - the original turtle serializer. It:

- uses PREFIX instead of @prefix
- uses BASE instead of @base
- adds a new line at RDF.type, or 'a'
- adds a newline and an indent for all triples with more than one object (object list)
- **adds a new line and ';' for the last triple in a set with '.'**
  on the start of the next line
- uses default encoding (encode()) is used instead of "latin-1"
- Nicholas Car, 2021

class rdflib.plugins.serializers.longturtle.**LongTurtleSerializer**(*store*)

Bases: *RecursiveSerializer*

__init__(*store*)

__module__ = 'rdflib.plugins.serializers.longturtle'

addNamespace(*prefix*, *namespace*)

base: Optional[str]

doList(*l_*)

encoding: str

endDocument()

getQName(*uri*, *gen_prefix=True*)

indentString = ' '

isValidList(*l_*)

Checks if l is a valid RDF list, i.e. no nodes have other properties.

label(*node*, *position*)

objectList(*objects*)

p_default(*node*, *position*, *newline=False*)

p_squared(*node*, *position*, *newline=False*)

path(*node*, *position*, *newline=False*)

predicateList(*subject*, *newline=False*)

preprocessTriple(*triple*)

reset()

s_default(*subject*)

s_squared(*subject*)

serialize(*stream*, *base=None*, *encoding=None*, *spacious=None*, ***args*)

Abstract method

short_name = 'longturtle'

startDocument()

statement(*subject*)

store: *Graph*

verb(*node*, *newline=False*)

rdflib.plugins.serializers.n3 module

Notation 3 (N3) RDF graph serializer for RDFLib.

class rdflib.plugins.serializers.n3.**N3Serializer**(*store*, *parent=None*)

Bases: *TurtleSerializer*

Parameters
store (*Graph*) –

__init__(*store*, *parent=None*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.n3'

base: Optional[str]

encoding: str

endDocument()

getQName(*uri*, *gen_prefix=True*)

indent(*modifier=0*)

Returns indent string multiplied by the depth

p_clause(*node*, *position*)

path(*node*, *position*, *newline=False*)

preprocessTriple(*triple*)

reset()

s_clause(*subject*)

short_name = 'n3'

statement(*subject*)

store: *Graph*

rdflib.plugins.serializers.nquads module

class rdflib.plugins.serializers.nquads.**NQuadsSerializer**(*store*)

Bases: *Serializer*

Parameters
store (*Graph*) –

__init__(*store*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.nquads'

base: Optional[str]

encoding: str

serialize(*stream*, *base=None*, *encoding=None*, ***args*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

store: *Graph*

rdflib.plugins.serializers.nt module

class rdflib.plugins.serializers.nt.**NTSerializer**(*store*)

Bases: *Serializer*

Serializes RDF graphs to NTriples format.

Parameters

store (*Graph*) –

__init__(*store*)

Parameters

store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.nt'

base: Optional[str]

encoding: str

serialize(*stream*, *base=None*, *encoding='utf-8'*, ***args*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

Return type

None

store: *Graph*

rdflib.plugins.serializers.rdfxml module

class rdflib.plugins.serializers.rdfxml.**PrettyXMLSerializer**(*store*, *max_depth=3*)

Bases: *Serializer*

Parameters
store (*Graph*) –

__init__(*store*, *max_depth=3*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.rdfxml'

base: Optional[str]

encoding: str

predicate(*predicate*, *object*, *depth=1*)

serialize(*stream*, *base=None*, *encoding=None*, ***args*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

store: *Graph*

subject(*subject*, *depth=1*)

Parameters

- **subject** (*IdentifiedNode*) –
- **depth** (int) –

class rdflib.plugins.serializers.rdfxml.**XMLSerializer**(*store*)

Bases: *Serializer*

Parameters
store (*Graph*) –

__init__(*store*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.rdfxml'

base: Optional[str]

encoding: str

predicate(*predicate*, *object*, *depth=1*)

serialize(*stream*, *base=None*, *encoding=None*, ***args*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

store: *Graph*

subject(*subject*, *depth=1*)

rdflib.plugins.serializers.rdfxml.**fix**(*val*)

strip off _: from nodeIDs... as they are not valid NCNames

rdflib.plugins.serializers.trig module

Trig RDF graph serializer for RDFLib. See <http://www.w3.org/TR/trig/> for syntax specification.

class rdflib.plugins.serializers.trig.**TrigSerializer**(*store*)

Bases: *TurtleSerializer*

Parameters

store (Union[*Graph*, *ConjunctiveGraph*]) –

__init__(*store*)

Parameters

store (Union[*Graph*, *ConjunctiveGraph*]) –

__module__ = 'rdflib.plugins.serializers.trig'

base: Optional[str]

encoding: str

indentString = ' '

preprocess()

Return type

None

reset()

Return type

None

serialize(*stream*, *base=None*, *encoding=None*, *spacious=None*, ***args*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

- **spacious** (Optional[bool]) –

short_name = 'trig'

store: *Graph*

rdflib.plugins.serializers.trix module

class rdflib.plugins.serializers.trix.**TriXSerializer**(*store*)

Bases: *Serializer*

Parameters
store (*Graph*) –

__init__(*store*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.plugins.serializers.trix'

base: Optional[str]

encoding: str

serialize(*stream*, *base=None*, *encoding=None*, ***args*)

Abstract method

Parameters

- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –

store: *Graph*

rdflib.plugins.serializers.turtle module

Turtle RDF graph serializer for RDFLib. See <http://www.w3.org/TeamSubmission/turtle/> for syntax specification.

class rdflib.plugins.serializers.turtle.**RecursiveSerializer**(*store*)

Bases: *Serializer*

__init__(*store*)

__module__ = 'rdflib.plugins.serializers.turtle'

addNamespace(*prefix*, *uri*)

base: Optional[str]

buildPredicateHash(*subject*)

Build a hash key by predicate to a list of objects for the given subject

checkSubject(*subject*)

Check to see if the subject should be serialized yet

encoding: *str*

indent(*modifier=0*)

Returns indent string multiplied by the depth

indentString = ' '

isDone(*subject*)

Return true if subject is serialized

maxDepth = 10

orderSubjects()

predicateOrder =
[rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#label')]

preprocess()

preprocessTriple(*spo*)

reset()

roundtrip_prefixes = ()

sortProperties(*properties*)

Take a hash from predicate uris to lists of values. Sort the lists of values. Return a sorted list of properties.

store: *Graph*

subjectDone(*subject*)

Mark a subject as done.

topClasses = [rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Class')]

write(*text*)

Write text in given encoding.

class rdflib.plugins.serializers.turtle.**TurtleSerializer**(*store*)

Bases: *RecursiveSerializer*

__init__(*store*)

__module__ = 'rdflib.plugins.serializers.turtle'

addNamespace(*prefix*, *namespace*)

base: Optional[str]

doList(*l_*)

encoding: *str*

endDocument()

getQName(*uri*, *gen_prefix=True*)

indentString = ' '

isValidList(*l_*)

Checks if l is a valid RDF list, i.e. no nodes have other properties.

label(*node*, *position*)

objectList(*objects*)

p_default(*node*, *position*, *newline=False*)

p_squared(*node*, *position*, *newline=False*)

path(*node*, *position*, *newline=False*)

predicateList(*subject*, *newline=False*)

preprocessTriple(*triple*)

reset()

s_default(*subject*)

s_squared(*subject*)

serialize(*stream*, *base=None*, *encoding=None*, *spacious=None*, ***args*)

Abstract method

short_name = 'turtle'

startDocument()

statement(*subject*)

store: *Graph*

verb(*node*, *newline=False*)

rdflib.plugins.serializers.xmlwriter module

class rdflib.plugins.serializers.xmlwriter.**XMLWriter**(*stream*, *namespace_manager*, *encoding=None*, *decl=1*, *extra_ns=None*)

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.serializers.xmlwriter', '__init__': <function XMLWriter.__init__>, '_XMLWriter__get_indent': <function XMLWriter.__get_indent>, 'indent': <property object>, '_XMLWriter__close_start_tag': <function XMLWriter.__close_start_tag>, 'push': <function XMLWriter.push>, 'pop': <function XMLWriter.pop>, 'element': <function XMLWriter.element>, 'namespaces': <function XMLWriter.namespaces>, 'attribute': <function XMLWriter.attribute>, 'text': <function XMLWriter.text>, 'qname': <function XMLWriter.qname>, '__dict__': <attribute '__dict__' of 'XMLWriter' objects>, '__weakref__': <attribute '__weakref__' of 'XMLWriter' objects>, '__doc__': None, '__annotations__': {}})

__init__(*stream*, *namespace_manager*, *encoding=None*, *decl=1*, *extra_ns=None*)

__module__ = 'rdflib.plugins.serializers.xmlwriter'

__weakref__
list of weak references to the object (if defined)

attribute(*uri*, *value*)

element(*uri*, *content*, *attributes={}*)
Utility method for adding a complete simple element

property indent

namespaces(*namespaces=None*)

pop(*uri=None*)

push(*uri*)

qname(*uri*)
Compute qname for a uri using our extra namespaces, or the given namespace manager

text(*text*)

Module contents

rdflib.plugins.shared package

Subpackages

rdflib.plugins.shared.jsonld package

Submodules

rdflib.plugins.shared.jsonld.context module

Implementation of the JSON-LD Context structure. See:

http://json-ld.org/

class rdflib.plugins.shared.jsonld.context.**Context**(*source=None*, *base=None*, *version=None*)
Bases: object

Parameters

- **source** (Optional[Any]) –
- **base** (Optional[str]) –
- **version** (Optional[float]) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.shared.jsonld.context', '__init__': <function Context.__init__>, 'base': <property object>, 'subcontext': <function Context.subcontext>, '_subcontext': <function Context._subcontext>, '_clear': <function Context._clear>, 'get_context_for_term': <function Context.get_context_for_term>, 'get_context_for_type': <function Context.get_context_for_type>, 'get_id': <function Context.get_id>, 'get_type': <function Context.get_type>, 'get_language': <function Context.get_language>, 'get_value': <function Context.get_value>, 'get_graph': <function Context.get_graph>, 'get_list': <function Context.get_list>, 'get_set': <function Context.get_set>, 'get_rev': <function Context.get_rev>, '_get': <function Context._get>, 'get_key': <function Context.get_key>, 'get_keys': <function Context.get_keys>, 'lang_key': <property object>, 'id_key': <property object>, 'type_key': <property object>, 'value_key': <property object>, 'list_key': <property object>, 'rev_key': <property object>, 'graph_key': <property object>, 'add_term': <function Context.add_term>, 'find_term': <function Context.find_term>, 'resolve': <function Context.resolve>, 'resolve_iri': <function Context.resolve_iri>, 'isblank': <function Context.isblank>, 'expand': <function Context.expand>, 'shrink_iri': <function Context.shrink_iri>, 'to_symbol': <function Context.to_symbol>, 'load': <function Context.load>, '_accept_term': <function Context._accept_term>, '_prep_sources': <function Context._prep_sources>, '_fetch_context': <function Context._fetch_context>, '_read_source': <function Context._read_source>, '_read_term': <function Context._read_term>, '_rec_expand': <function Context._rec_expand>, '_prep_expand': <function Context._prep_expand>, '_get_source_id': <function Context._get_source_id>, '__dict__': <attribute '__dict__' of 'Context' objects>, '__weakref__': <attribute '__weakref__' of 'Context' objects>, '__doc__': None, '__annotations__': {'version': 'float', 'vocab': 'Optional[str]', '_base': 'Optional[str]', 'terms': 'Dict[str, Any]', '_alias': 'Dict[str, List[str]]', '_lookup': 'Dict[Tuple[str, Any, Union[Defined, str], bool], Any]', '_prefixes': 'Dict[str, Any]', 'parent': 'Optional[Context]', '_context_cache': 'Dict[str, Any]'}})

__init__(*source=None*, *base=None*, *version=None*)

Parameters

- **source** (Optional[Any]) –
- **base** (Optional[str]) –
- **version** (Optional[float]) –

__module__ = 'rdflib.plugins.shared.jsonld.context'

__weakref__

list of weak references to the object (if defined)

add_term(*name*, *idref*, *coercion=0*, *container=0*, *index=None*, *language=0*, *reverse=False*, *context=0*, *prefix=None*, *protected=False*)

Parameters

- **name** (str) –
- **idref** (str) –
- **coercion** (Union[*Defined*, str]) –
- **container** (Union[Collection[Any], str, *Defined*]) –

- **index** (Union[str, *Defined*, None]) –
- **language** (Union[str, *Defined*, None]) –
- **reverse** (bool) –
- **context** (Any) –
- **prefix** (Optional[bool]) –
- **protected** (bool) –

property base: Optional[str]

Return type
Optional[str]

expand(*term_curie_or_iri*, *use_vocab=True*)

Parameters

- **term_curie_or_iri** (Any) –
- **use_vocab** (bool) –

Return type
Optional[str]

find_term(*idref*, *coercion=None*, *container=0*, *language=None*, *reverse=False*)

Parameters

- **idref** (str) –
- **coercion** (Union[str, *Defined*, None]) –
- **container** (Union[*Defined*, str]) –
- **language** (Optional[str]) –
- **reverse** (bool) –

get_context_for_term(*term*)

Parameters
term (Optional[*Term*]) –

Return type
Context

get_context_for_type(*node*)

Parameters
node (Any) –

Return type
Optional[*Context*]

get_graph(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

`get_id`(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

`get_key`(*key*)

Parameters
key (str) –

Return type
str

`get_keys`(*key*)

Parameters
key (str) –

Return type
Generator[str, None, None]

`get_language`(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

`get_list`(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

`get_rev`(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

`get_set`(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

`get_type`(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

get_value(*obj*)

Parameters
obj (Dict[str, Any]) –

Return type
Any

property graph_key

property id_key

isblank(*ref*)

Parameters
ref (str) –

Return type
bool

property lang_key

property list_key

load(*source*, *base=None*, *referenced_contexts=None*)

Parameters

- **source** (Union[List[Any], Any, None]) –
- **base** (Optional[str]) –
- **referenced_contexts** (Optional[Set[Any]]) –

resolve(*curie_or_iri*)

Parameters
curie_or_iri (str) –

Return type
str

resolve_iri(*iri*)

Parameters
iri (str) –

Return type
str

property rev_key

shrink_iri(*iri*)

Parameters
iri (str) –

Return type
str

subcontext(*source*, *propagate=True*)

Parameters

- **source** (Any) –

- **propagate** (bool) –

Return type
Context

to_symbol(*iri*)

Parameters
iri (str) –

Return type
Optional[str]

property type_key

property value_key

class rdflib.plugins.shared.jsonld.context.**Defined**

Bases: int

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.shared.jsonld.context', '__dict__': <attribute '__dict__' of 'Defined' objects>, '__doc__': None, '__annotations__': {}})

__module__ = 'rdflib.plugins.shared.jsonld.context'

class rdflib.plugins.shared.jsonld.context.**Term**(*id*, *name*, *type*, *container*, *index*, *language*, *reverse*, *context*, *prefix*, *protected*)

Bases: tuple

__getnewargs__()

Return self as a plain tuple. Used by copy and pickle.

__module__ = 'rdflib.plugins.shared.jsonld.context'

static __new__(*_cls*, *id*, *name*, *type=0*, *container=0*, *index=0*, *language=0*, *reverse=False*, *context=0*, *prefix=False*, *protected=False*)

Create new instance of Term(id, name, type, container, index, language, reverse, context, prefix, protected)

__repr__()

Return a nicely formatted representation string

__slots__ = ()

container

Alias for field number 3

context

Alias for field number 7

id

Alias for field number 0

index

Alias for field number 4

language

Alias for field number 5

name

Alias for field number 1

prefix

Alias for field number 8

protected

Alias for field number 9

reverse

Alias for field number 6

type

Alias for field number 2

rdflib.plugins.shared.jsonld.errors module

exception rdflib.plugins.shared.jsonld.errors.**JSONLDException**

Bases: ValueError

__module__ = 'rdflib.plugins.shared.jsonld.errors'

__weakref__

list of weak references to the object (if defined)

rdflib.plugins.shared.jsonld.keys module

rdflib.plugins.shared.jsonld.util module

rdflib.plugins.shared.jsonld.util.**context_from_urlinputsource**(*source*)

Please note that JSON-LD documents served with the application/ld+json media type MUST have all context information, including references to external contexts, within the body of the document. Contexts linked via a http://www.w3.org/ns/json-ld#context HTTP Link Header MUST be ignored for such documents.

Parameters

source (*URLInputSource*) –

Return type

Optional[str]

rdflib.plugins.shared.jsonld.util.**norm_url**(*base*, *url*)

```
>>> norm_url('http://example.org/', '/one')
'http://example.org/one'
>>> norm_url('http://example.org/', '/one#')
'http://example.org/one#'
>>> norm_url('http://example.org/one', 'two')
'http://example.org/two'
>>> norm_url('http://example.org/one/', 'two')
'http://example.org/one/two'
>>> norm_url('http://example.org/', 'http://example.net/one')
'http://example.net/one'
>>> norm_url('http://example.org/', 'http://example.org//one')
'http://example.org//one'
```

Parameters

- **base** (str) –
- **url** (str) –

Return type
str

rdflib.plugins.shared.jsonld.util.**source_to_json**(*source*)

Parameters
source (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –

Return type
Optional[Any]

rdflib.plugins.shared.jsonld.util.**split_iri**(*iri*)

Parameters
iri (str) –

Return type
Tuple[str, Optional[str]]

Module contents

Module contents

rdflib.plugins.sparql package

Subpackages

rdflib.plugins.sparql.results package

Submodules

rdflib.plugins.sparql.results.csvresults module

class rdflib.plugins.sparql.results.csvresults.**CSVResultParser**

Bases: *ResultParser*

__init__()

__module__ = 'rdflib.plugins.sparql.results.csvresults'

convertTerm(*t*)

Parameters
t (str) –

Return type
Union[*BNode*, *URIRef*, *Literal*, None]

parse(*source*, *content_type=None*)

return a Result object

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

Return type
Result

parseRow(*row*, *v*)

Parameters

- **row** (List[str]) –
- **v** (List[*Variable*]) –

Return type
Dict[*Variable*, Union[*BNode*, *URIRef*, *Literal*]]

class rdflib.plugins.sparql.results.csvresults.**CSVResultSerializer**(*result*)

Bases: *ResultSerializer*

Parameters
result (*SPARQLResult*) –

__init__(*result*)

Parameters
result (*SPARQLResult*) –

__module__ = 'rdflib.plugins.sparql.results.csvresults'

serialize(*stream*, *encoding='utf-8'*, ***kwargs*)

return a string properly serialized

Parameters

- **stream** (IO) –
- **encoding** (str) –

Return type
None

serializeTerm(*term*, *encoding*)

Parameters

- **term** (Optional[*Identifier*]) –
- **encoding** (str) –

Return type
Union[str, *Identifier*]

rdflib.plugins.sparql.results.graph module

class rdflib.plugins.sparql.results.graph.**GraphResultParser**

Bases: *ResultParser*

__module__ = 'rdflib.plugins.sparql.results.graph'

parse(*source*, *content_type*)

return a Result object

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

Return type

Result

rdflib.plugins.sparql.results.jsonresults module

class rdflib.plugins.sparql.results.jsonresults.**JSONResult**(*json*)

Bases: *Result*

Parameters

json (Dict[str, Any]) –

__init__(*json*)

Parameters

json (Dict[str, Any]) –

__module__ = 'rdflib.plugins.sparql.results.jsonresults'

askAnswer: Optional[bool]

graph: Optional['Graph']

vars: Optional[List['Variable']]

variables contained in the result.

class rdflib.plugins.sparql.results.jsonresults.**JSONResultParser**

Bases: *ResultParser*

__module__ = 'rdflib.plugins.sparql.results.jsonresults'

parse(*source*, *content_type=None*)

return a Result object

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

Return type

Result

class rdflib.plugins.sparql.results.jsonresults.**JSONResultSerializer**(*result*)

Bases: *ResultSerializer*

Parameters
result (*Result*) –

__init__(*result*)

Parameters
result (*Result*) –

__module__ = 'rdflib.plugins.sparql.results.jsonresults'

serialize(*stream*, *encoding=None*)

return a string properly serialized

Parameters

- **stream** (IO) –
- **encoding** (Optional[str]) –

Return type
None

rdflib.plugins.sparql.results.jsonresults.**parseJsonTerm**(*d*)

rdflib object (Literal, URIRef, BNode) for the given json-format dict.

input is like:
{ 'type': 'uri', 'value': 'http://famegame.com/2006/01/username' } { 'type': 'literal', 'value': 'drewp' }

Parameters
d (Dict[str, str]) –

Return type
Identifier

rdflib.plugins.sparql.results.jsonresults.**termToJSON**(*self*, *term*)

Parameters

- **self** (*JSONResultSerializer*) –
- **term** (Optional[*Identifier*]) –

Return type
Optional[Dict[str, str]]

rdflib.plugins.sparql.results.rdfresults module

class rdflib.plugins.sparql.results.rdfresults.**RDFResult**(*source*, ***kwargs*)

Bases: *Result*

Parameters

- **source** (Union[IO, *Graph*]) –
- **kwargs** (Any) –

__init__(*source*, ***kwargs*)

Parameters

- **source** (Union[IO, *Graph*]) –
- **kwargs** (Any) –

__module__ = 'rdflib.plugins.sparql.results.rdfresults'

askAnswer: Optional[bool]

graph: Optional['Graph']

vars: Optional[List['Variable']]

variables contained in the result.

class rdflib.plugins.sparql.results.rdfresults.**RDFResultParser**

Bases: *ResultParser*

__module__ = 'rdflib.plugins.sparql.results.rdfresults'

parse(*source*, ***kwargs*)

return a Result object

Parameters

- **source** (Union[IO, *Graph*]) –
- **kwargs** (Any) –

Return type

Result

rdflib.plugins.sparql.results.tsvresults module

This implements the Tab Separated SPARQL Result Format

It is implemented with pyparsing, reusing the elements from the SPARQL Parser

class rdflib.plugins.sparql.results.tsvresults.**TSVResultParser**

Bases: *ResultParser*

__module__ = 'rdflib.plugins.sparql.results.tsvresults'

convertTerm(*t*)

Parameters

t (Union[object, *Literal*, *BNode*, *CompValue*, *URIRef*]) –

Return type

Union[object, *BNode*, *URIRef*, *Literal*, None]

parse(*source*, *content_type=None*)

return a Result object

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

Return type
Result

rdflib.plugins.sparql.results.txtresults module

class rdflib.plugins.sparql.results.txtresults.**TXTResultSerializer**(*result*)

Bases: *ResultSerializer*

A write only QueryResult serializer for text/ascii tables

Parameters
result (*Result*) –

__module__ = 'rdflib.plugins.sparql.results.txtresults'

serialize(*stream*, *encoding*, *namespace_manager=None*)

return a text table of query results

Parameters

- **stream** (IO) –
- **encoding** (str) –
- **namespace_manager** (Optional[*NamespaceManager*]) –

Return type
None

rdflib.plugins.sparql.results.xmlresults module

class rdflib.plugins.sparql.results.xmlresults.**SPARQLXMLWriter**(*output*, *encoding='utf-8'*)

Bases: object

Python saxutils-based SPARQL XML Writer

Parameters

- **output** (IO) –
- **encoding** (str) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.results.xmlresults', '__doc__': '\n Python saxutils-based SPARQL XML Writer\n ', '__init__': <function SPARQLXMLWriter.__init__>, 'write_header': <function SPARQLXMLWriter.write_header>, 'write_ask': <function SPARQLXMLWriter.write_ask>, 'write_results_header': <function SPARQLXMLWriter.write_results_header>, 'write_start_result': <function SPARQLXMLWriter.write_start_result>, 'write_end_result': <function SPARQLXMLWriter.write_end_result>, 'write_binding': <function SPARQLXMLWriter.write_binding>, 'close': <function SPARQLXMLWriter.close>, '__dict__': <attribute '__dict__' of 'SPARQLXMLWriter' objects>, '__weakref__': <attribute '__weakref__' of 'SPARQLXMLWriter' objects>, '__annotations__': {}})

__init__(*output*, *encoding='utf-8'*)

Parameters

- **output** (IO) –

- **encoding** (str) –

__module__ = 'rdflib.plugins.sparql.results.xmlresults'

__weakref__
: list of weak references to the object (if defined)

close()

Return type
None

write_ask(*val*)

Parameters
val (bool) –

Return type
None

write_binding(*name*, *val*)

Parameters

- **name** (*Variable*) –
- **val** (*Identifier*) –

Return type
None

write_end_result()

Return type
None

write_header(*allvarsL*)

Parameters
allvarsL (Sequence[*Variable*]) –

Return type
None

write_results_header()

Return type
None

write_start_result()

Return type
None

class rdflib.plugins.sparql.results.xmlresults.**XMLResult**(*source*, *content_type=None*)

Bases: *Result*

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

__init__(*source*, *content_type=None*)

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

__module__ = 'rdflib.plugins.sparql.results.xmlresults'

askAnswer: Optional[bool]

graph: Optional['Graph']

vars: Optional[List['Variable']]

variables contained in the result.

class rdflib.plugins.sparql.results.xmlresults.**XMLResultParser**

Bases: *ResultParser*

__module__ = 'rdflib.plugins.sparql.results.xmlresults'

parse(*source*, *content_type=None*)

return a Result object

Parameters

- **source** (IO) –
- **content_type** (Optional[str]) –

Return type

Result

class rdflib.plugins.sparql.results.xmlresults.**XMLResultSerializer**(*result*)

Bases: *ResultSerializer*

Parameters

result (*Result*) –

__init__(*result*)

Parameters

result (*Result*) –

__module__ = 'rdflib.plugins.sparql.results.xmlresults'

serialize(*stream*, *encoding='utf-8'*, ***kwargs*)

return a string properly serialized

Parameters

- **stream** (IO) –
- **encoding** (str) –
- **kwargs** (Any) –

Return type

None

rdflib.plugins.sparql.results.xmlresults.**log = <Logger rdflib.plugins.sparql.results.xmlresults (WARNING)>**

A Parser for SPARQL results in XML:

http://www.w3.org/TR/rdf-sparql-XMLres/

Bits and pieces borrowed from: http://projects.bigasterisk.com/sparqlhttp/

Authors: Drew Perttula, Gunnar Aastrand Grimnes

rdflib.plugins.sparql.results.xmlresults.**parseTerm**(*element*)

rdflib object (Literal, URIRef, BNode) for the given elementtree element

Parameters
element (Element) –

Return type
Union[*URIRef*, *Literal*, *BNode*]

Module contents

Parsers and serializers for SPARQL Result formats

Submodules

rdflib.plugins.sparql.aggregates module

class rdflib.plugins.sparql.aggregates.**Accumulator**(*aggregation*)

Bases: object

abstract base class for different aggregation functions

Parameters
aggregation (*CompValue*) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.aggregates', '__doc__': 'abstract base class for different aggregation functions', '__init__': <function Accumulator.__init__>, 'dont_care': <function Accumulator.dont_care>, 'use_row': <function Accumulator.use_row>, 'set_value': <function Accumulator.set_value>, '__dict__': <attribute '__dict__' of 'Accumulator' objects>, '__weakref__': <attribute '__weakref__' of 'Accumulator' objects>, '__annotations__': {'get_value': 'Callable[[], Optional[Literal]]', 'update': "Callable[[FrozenBindings, 'Aggregator'], None]", 'seen': 'Set[Any]'}})

__init__(*aggregation*)

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

__weakref__

list of weak references to the object (if defined)

dont_care(*row*)

skips distinct test

Parameters
row (*FrozenBindings*) –

Return type
bool

set_value(*bindings*)

sets final value in bindings

Parameters
bindings (MutableMapping[*Variable*, *Identifier*]) –

Return type
None

use_row(*row*)

tests distinct with set

Parameters
row (*FrozenBindings*) –

Return type
bool

class rdflib.plugins.sparql.aggregates.**Aggregator**(*aggregations*)

Bases: object

combines different Accumulator objects

Parameters
aggregations (List[*CompValue*]) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.aggregates', '__doc__': 'combines different Accumulator objects', 'accumulator_classes': {'Aggregate_Count': <class 'rdflib.plugins.sparql.aggregates.Counter'>, 'Aggregate_Sample': <class 'rdflib.plugins.sparql.aggregates.Sample'>, 'Aggregate_Sum': <class 'rdflib.plugins.sparql.aggregates.Sum'>, 'Aggregate_Avg': <class 'rdflib.plugins.sparql.aggregates.Average'>, 'Aggregate_Min': <class 'rdflib.plugins.sparql.aggregates.Minimum'>, 'Aggregate_Max': <class 'rdflib.plugins.sparql.aggregates.Maximum'>, 'Aggregate_GroupConcat': <class 'rdflib.plugins.sparql.aggregates.GroupConcat'>}, '__init__': <function Aggregator.__init__>, 'update': <function Aggregator.update>, 'get_bindings': <function Aggregator.get_bindings>, '__dict__': <attribute '__dict__' of 'Aggregator' objects>, '__weakref__': <attribute '__weakref__' of 'Aggregator' objects>, '__annotations__': {'bindings': 'Dict[Variable, Identifier]', 'accumulators': 'Dict[str, Accumulator]'}})

__init__(*aggregations*)

Parameters
aggregations (List[*CompValue*]) –

__module__ = 'rdflib.plugins.sparql.aggregates'

__weakref__

list of weak references to the object (if defined)

accumulator_classes = {'Aggregate_Avg': <class 'rdflib.plugins.sparql.aggregates.Average'>, 'Aggregate_Count': <class 'rdflib.plugins.sparql.aggregates.Counter'>, 'Aggregate_GroupConcat': <class 'rdflib.plugins.sparql.aggregates.GroupConcat'>, 'Aggregate_Max': <class 'rdflib.plugins.sparql.aggregates.Maximum'>, 'Aggregate_Min': <class 'rdflib.plugins.sparql.aggregates.Minimum'>, 'Aggregate_Sample': <class 'rdflib.plugins.sparql.aggregates.Sample'>, 'Aggregate_Sum': <class 'rdflib.plugins.sparql.aggregates.Sum'>}

get_bindings()

calculate and set last values

Return type
Mapping[*Variable*, *Identifier*]

update(*row*)

update all own accumulators

Parameters
row (*FrozenBindings*) –

Return type
None

class rdflib.plugins.sparql.aggregates.**Average**(*aggregation*)

Bases: *Accumulator*

Parameters
aggregation (*CompValue*) –

__init__(*aggregation*)

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

get_value()

Return type
Literal

seen: Set[Any]

update(*row*, *aggregator*)

Parameters

- **row** (*FrozenBindings*) –
- **aggregator** (*Aggregator*) –

Return type
None

class rdflib.plugins.sparql.aggregates.**Counter**(*aggregation*)

Bases: *Accumulator*

Parameters
aggregation (*CompValue*) –

__init__(*aggregation*)

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

eval_full_row(*row*)

Parameters
row (*FrozenBindings*) –

Return type
FrozenBindings

eval_row(*row*)

Parameters
row (*FrozenBindings*) –

Return type
Identifier

get_value()

Return type
Literal

seen: Set[Any]

update(*row*, *aggregator*)

Parameters

- **row** (*FrozenBindings*) –
- **aggregator** (*Aggregator*) –

Return type
None

use_row(*row*)

tests distinct with set

Parameters
row (*FrozenBindings*) –

Return type
bool

class rdflib.plugins.sparql.aggregates.**Extremum**(*aggregation*)

Bases: *Accumulator*

abstract base class for Minimum and Maximum

Parameters
aggregation (*CompValue*) –

__init__(*aggregation*)

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

get_value: Callable[[], Optional[Literal]]

seen: Set[Any]

set_value(*bindings*)

sets final value in bindings

Parameters
bindings (MutableMapping[*Variable*, *Identifier*]) –

Return type
None

update(*row*, *aggregator*)

Parameters

- **row** (*FrozenBindings*) –
- **aggregator** (*Aggregator*) –

Return type
None

class rdflib.plugins.sparql.aggregates.**GroupConcat**(*aggregation*)

Bases: *Accumulator*

__init__(*aggregation*)

__module__ = 'rdflib.plugins.sparql.aggregates'

get_value()

Return type
Literal

seen: Set[Any]

update(*row*, *aggregator*)

Parameters

- **row** (*FrozenBindings*) –
- **aggregator** (*Aggregator*) –

Return type
None

class rdflib.plugins.sparql.aggregates.**Maximum**(*aggregation*)

Bases: *Extremum*

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

compare(*val1*, *val2*)

Parameters

- **val1** (TypeVar(_ValueT, *Variable*, *BNode*, *URIRef*, *Literal*)) –

- **val2** (TypeVar(_ValueT, *Variable*, *BNode*, *URIRef*, *Literal*)) –

Return type
TypeVar(_ValueT, *Variable*, *BNode*, *URIRef*, *Literal*)

get_value: Callable[[], Optional[*Literal*]]

seen: Set[Any]

value: Any

class rdflib.plugins.sparql.aggregates.**Minimum**(*aggregation*)

Bases: *Extremum*

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

compare(*val1*, *val2*)

Parameters

- **val1** (TypeVar(_ValueT, *Variable*, *BNode*, *URIRef*, *Literal*)) –
- **val2** (TypeVar(_ValueT, *Variable*, *BNode*, *URIRef*, *Literal*)) –

Return type
TypeVar(_ValueT, *Variable*, *BNode*, *URIRef*, *Literal*)

get_value: Callable[[], Optional[*Literal*]]

seen: Set[Any]

value: Any

class rdflib.plugins.sparql.aggregates.**Sample**(*aggregation*)

Bases: *Accumulator*

takes the first eligible value

__init__(*aggregation*)

__module__ = 'rdflib.plugins.sparql.aggregates'

get_value()

Return type
None

seen: Set[Any]

update(*row*, *aggregator*)

Parameters

- **row** (*FrozenBindings*) –
- **aggregator** (*Aggregator*) –

Return type
None

class rdflib.plugins.sparql.aggregates.**Sum**(*aggregation*)

Bases: *Accumulator*

Parameters
aggregation (*CompValue*) –

__init__(*aggregation*)

Parameters
aggregation (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.aggregates'

get_value()

Return type
Literal

seen: Set[Any]

update(*row*, *aggregator*)

Parameters

- **row** (*FrozenBindings*) –
- **aggregator** (*Aggregator*) –

Return type
None

rdflib.plugins.sparql.aggregates.**type_safe_numbers**(**args: int*) → Tuple[int]

rdflib.plugins.sparql.aggregates.**type_safe_numbers**(**args: Union[Decimal, float, int]*) → Tuple[Union[float, int]]

Parameters
args (Union[Decimal, float, int]) –

Return type
Iterable[Union[float, int]]

rdflib.plugins.sparql.algebra module

rdflib.plugins.sparql.algebra.**BGP**(*triples=None*)

Parameters
triples (Optional[List[Tuple[*Identifier*, *Identifier*, *Identifier*]]]) –

Return type
CompValue

exception rdflib.plugins.sparql.algebra.**ExpressionNotCoveredException**

Bases: Exception

__module__ = 'rdflib.plugins.sparql.algebra'

__weakref__
list of weak references to the object (if defined)

rdflib.plugins.sparql.algebra.**Extend**(*p*, *expr*, *var*)

Parameters

- **p** (*CompValue*) –
- **expr** (Union[*Identifier*, *Expr*]) –
- **var** (*Variable*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Filter**(*expr*, *p*)

Parameters

- **expr** (*Expr*) –
- **p** (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Graph**(*term*, *graph*)

Parameters

- **term** (*Identifier*) –
- **graph** (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Group**(*p*, *expr=None*)

Parameters

- **p** (*CompValue*) –
- **expr** (Optional[List[*Variable*]]) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Join**(*p1*, *p2*)

Parameters

- **p1** (*CompValue*) –
- **p2** (Optional[*CompValue*]) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**LeftJoin**(*p1*, *p2*, *expr*)

Parameters

- **p1** (*CompValue*) –
- **p2** (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Minus**(*p1*, *p2*)

Parameters

- **p1** (*CompValue*) –
- **p2** (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**OrderBy**(*p*, *expr*)

Parameters

- **p** (*CompValue*) –
- **expr** (List[*CompValue*]) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Project**(*p*, *PV*)

Parameters

- **p** (*CompValue*) –
- **PV** (List[*Variable*]) –

Return type
CompValue

exception rdflib.plugins.sparql.algebra.**StopTraversal**(*rv*)

Bases: Exception

Parameters
rv (bool) –

__init__(*rv*)

Parameters
rv (bool) –

__module__ = 'rdflib.plugins.sparql.algebra'

__weakref__
list of weak references to the object (if defined)

rdflib.plugins.sparql.algebra.**ToMultiSet**(*p*)

Parameters
p (Union[List[Dict[*Variable*, str]], *CompValue*]) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Union**(*p1*, *p2*)

Parameters

- **p1** (*CompValue*) –
- **p2** (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**Values**(*res*)

Parameters
res (List[Dict[*Variable*, str]]) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**analyse**(*n*, *children*)

Some things can be lazily joined. This propegates whether they can up the tree and sets lazy flags for all joins

Parameters

- **n** (Any) –
- **children** (Any) –

Return type
bool

rdflib.plugins.sparql.algebra.**collectAndRemoveFilters**(*parts*)

FILTER expressions apply to the whole group graph pattern in which they appear.

http://www.w3.org/TR/sparql11-query/#sparqlCollectFilters

Parameters
parts (List[*CompValue*]) –

Return type
Optional[*Expr*]

rdflib.plugins.sparql.algebra.**pprintAlgebra**(*q*)

Return type
None

rdflib.plugins.sparql.algebra.**reorderTriples**(*l_*)

Reorder triple patterns so that we execute the ones with most bindings first

Parameters
l_ (Iterable[Tuple[*Identifier*, *Identifier*, *Identifier*]]) –

Return type
List[Tuple[*Identifier*, *Identifier*, *Identifier*]]

rdflib.plugins.sparql.algebra.**simplify**(*n*)

Remove joins to empty BGPs

Parameters
n (Any) –

Return type
Optional[*CompValue*]

rdflib.plugins.sparql.algebra.**translate**(*q*)

http://www.w3.org/TR/sparql11-query/#convertSolMod

Parameters
q (*CompValue*) –

Return type
Tuple[Optional[*CompValue*], List[*Variable*]]

rdflib.plugins.sparql.algebra.**translateAggregates**(*q*, *M*)

Parameters

- **q** (*CompValue*) –
- **M** (*CompValue*) –

Return type
Tuple[*CompValue*, List[Tuple[*Variable*, *Variable*]]]

rdflib.plugins.sparql.algebra.**translateAlgebra**(*query_algebra*)

Parameters
query_algebra (*Query*) – An algebra returned by the function call algebra.translateQuery(parse_tree).

Return type
str

Returns
The query form generated from the SPARQL 1.1 algebra tree for select queries.

rdflib.plugins.sparql.algebra.**translateExists**(*e*)

Translate the graph pattern used by EXISTS and NOT EXISTS http://www.w3.org/TR/sparql11-query/#sparqlCollectFilters

Parameters
e (Union[*Expr*, *Literal*, *Variable*, *URIRef*]) –

Return type
Union[*Expr*, *Literal*, *Variable*, *URIRef*]

rdflib.plugins.sparql.algebra.**translateGraphGraphPattern**(*graphPattern*)

Parameters
graphPattern (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**translateGroupGraphPattern**(*graphPattern*)

http://www.w3.org/TR/sparql11-query/#convertGraphPattern

Parameters
graphPattern (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**translateGroupOrUnionGraphPattern**(*graphPattern*)

Parameters
graphPattern (*CompValue*) –

Return type
Optional[*CompValue*]

rdflib.plugins.sparql.algebra.**translateInlineData**(*graphPattern*)

Parameters
graphPattern (*CompValue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**translatePName**(*p*, *prologue*)

Expand prefixed/relative URIs

Parameters

- **p** (Union[*CompValue*, str]) –
- **prologue** (*Prologue*) –

Return type
Optional[*Identifier*]

rdflib.plugins.sparql.algebra.**translatePath**(*p:* URIRef) → None

rdflib.plugins.sparql.algebra.**translatePath**(*p:* CompValue) → *Path*

Translate PropertyPath expressions

Parameters
p (Union[*CompValue*, *URIRef*]) –

Return type
Optional[*Path*]

rdflib.plugins.sparql.algebra.**translatePrologue**(*p*, *base*, *initNs=None*, *prologue=None*)

Parameters

- **p** (ParseResults) –
- **base** (Optional[str]) –
- **initNs** (Optional[Mapping[str, Any]]) –
- **prologue** (Optional[*Prologue*]) –

Return type
Prologue

rdflib.plugins.sparql.algebra.**translateQuads**(*quads*)

Parameters
quads (*CompValue*) –

Return type
Tuple[List[Tuple[*Identifier*, *Identifier*, *Identifier*]], DefaultDict[str, List[Tuple[*Identifier*, *Identifier*, *Identifier*]]]]

rdflib.plugins.sparql.algebra.**translateQuery**(*q*, *base=None*, *initNs=None*)

Translate a query-parsetree to a SPARQL Algebra Expression

Return a rdflib.plugins.sparql.sparql.Query object

Parameters

- **q** (ParseResults) –
- **base** (Optional[str]) –

- **initNs** (Optional[Mapping[str, Any]]) –

Return type
Query

rdflib.plugins.sparql.algebra.**translateUpdate**(*q*, *base=None*, *initNs=None*)

Returns a list of SPARQL Update Algebra expressions

Parameters

- **q** (*CompValue*) –
- **base** (Optional[str]) –
- **initNs** (Optional[Mapping[str, Any]]) –

Return type
Update

rdflib.plugins.sparql.algebra.**translateUpdate1**(*u*, *prologue*)

Parameters

- **u** (*CompValue*) –
- **prologue** (*Prologue*) –

Return type
CompValue

rdflib.plugins.sparql.algebra.**translateValues**(*v*)

Parameters
v (*CompValue*) –

Return type
Union[List[Dict[*Variable*, str]], *CompValue*]

rdflib.plugins.sparql.algebra.**traverse**(*tree*, *visitPre=<function <lambda>>*, *visitPost=<function <lambda>>*, *complete=None*)

Traverse tree, visit each node with visit function visit function may raise StopTraversal to stop traversal if complete!=None, it is returned on complete traversal, otherwise the transformed tree is returned

Parameters

- **visitPre** (Callable[[Any], Any]) –
- **visitPost** (Callable[[Any], Any]) –
- **complete** (Optional[bool]) –

Return type
Any

rdflib.plugins.sparql.algebra.**triples**(*l*)

Parameters
l (Union[List[List[*Identifier*]], List[Tuple[*Identifier*, *Identifier*, *Identifier*]]]) –

Return type
List[Tuple[*Identifier*, *Identifier*, *Identifier*]]

rdflib.plugins.sparql.datatypes module

rdflib.plugins.sparql.datatypes.**type_promotion**(*t1*, *t2*)

Parameters

- **t1** (*URIRef*) –
- **t2** (Optional[*URIRef*]) –

Return type
URIRef

rdflib.plugins.sparql.evaluate module

rdflib.plugins.sparql.evaluate.**evalAggregateJoin**(*ctx*, *agg*)

Parameters

- **ctx** (*QueryContext*) –
- **agg** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalAskQuery**(*ctx*, *query*)

Parameters

- **ctx** (*QueryContext*) –
- **query** (*CompValue*) –

Return type
Mapping[str, Union[str, bool]]

rdflib.plugins.sparql.evaluate.**evalBGP**(*ctx*, *bgp*)

A basic graph pattern

Parameters

- **ctx** (*QueryContext*) –
- **bgp** (List[Tuple[*Identifier*, *Identifier*, *Identifier*]]) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalConstructQuery**(*ctx*, *query*)

Parameters

- **ctx** (*QueryContext*) –
- **query** (*CompValue*) –

Return type
Mapping[str, Union[str, *Graph*]]

rdflib.plugins.sparql.evaluate.**evalDescribeQuery**(*ctx*, *query*)

Parameters
ctx (*QueryContext*) –

Return type
Dict[str, Union[str, *Graph*]]

rdflib.plugins.sparql.evaluate.**evalDistinct**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalExtend**(*ctx*, *extend*)

Parameters

- **ctx** (*QueryContext*) –
- **extend** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalFilter**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalGraph**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalGroup**(*ctx*, *group*)
http://www.w3.org/TR/sparql11-query/#defn_algGroup

Parameters

- **ctx** (*QueryContext*) –
- **group** (*CompValue*) –

rdflib.plugins.sparql.evaluate.**evalJoin**(*ctx*, *join*)

Parameters

- **ctx** (*QueryContext*) –

- **join** (*CompValue*) –

Return type
Generator[*FrozenDict*, None, None]

rdflib.plugins.sparql.evaluate.**evalLazyJoin**(*ctx*, *join*)

A lazy join will push the variables bound in the first part to the second part, essentially doing the join implicitly hopefully evaluating much fewer triples

Parameters

- **ctx** (*QueryContext*) –
- **join** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalLeftJoin**(*ctx*, *join*)

Parameters

- **ctx** (*QueryContext*) –
- **join** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalMinus**(*ctx*, *minus*)

Parameters

- **ctx** (*QueryContext*) –
- **minus** (*CompValue*) –

Return type
Generator[*FrozenDict*, None, None]

rdflib.plugins.sparql.evaluate.**evalMultiset**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

rdflib.plugins.sparql.evaluate.**evalOrderBy**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalPart**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Any

rdflib.plugins.sparql.evaluate.**evalProject**(*ctx*, *project*)

Parameters

- **ctx** (*QueryContext*) –
- **project** (*CompValue*) –

rdflib.plugins.sparql.evaluate.**evalQuery**(*graph*, *query*, *initBindings*, *base=None*)

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in `SERVICE` directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **graph** (*Graph*) –
- **query** (*Query*) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **base** (Optional[str]) –

Return type
Mapping[Any, Any]

rdflib.plugins.sparql.evaluate.**evalReduced**(*ctx*, *part*)

apply REDUCED to result

REDUCED is not as strict as DISTINCT, but if the incoming rows were sorted it should produce the same result with limited extra memory and time per incoming row.

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evaluate.**evalSelectQuery**(*ctx*, *query*)

Parameters

- **ctx** (*QueryContext*) –
- **query** (*CompValue*) –

Return type
Mapping[str, Union[str, List[*Variable*], Iterable[*FrozenDict*]]]

rdflib.plugins.sparql.evaluate.**evalServiceQuery**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

rdflib.plugins.sparql.evaluate.**evalSlice**(*ctx*, *slice*)

Parameters

- **ctx** (*QueryContext*) –
- **slice** (*CompValue*) –

rdflib.plugins.sparql.evaluate.**evalUnion**(*ctx*, *union*)

Parameters

- **ctx** (*QueryContext*) –
- **union** (*CompValue*) –

Return type
Iterable[*FrozenBindings*]

rdflib.plugins.sparql.evaluate.**evalValues**(*ctx*, *part*)

Parameters

- **ctx** (*QueryContext*) –
- **part** (*CompValue*) –

Return type
Generator[*FrozenBindings*, None, None]

rdflib.plugins.sparql.evalutils module

rdflib.plugins.sparql.operators module

rdflib.plugins.sparql.operators.**AdditiveExpression**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (Union[*QueryContext*, *FrozenBindings*]) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_ABS**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-abs

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_BNODE**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-bnode

Return type
BNode

rdflib.plugins.sparql.operators.**Builtin_BOUND**(*e*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-bound

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_CEIL**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-ceil

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_COALESCE**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-coalesce

Parameters
expr (*Expr*) –

rdflib.plugins.sparql.operators.**Builtin_CONCAT**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-concat

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_CONTAINS**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strcontains

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_DATATYPE**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Optional[str]

rdflib.plugins.sparql.operators.**Builtin_DAY**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_ENCODE_FOR_URI**(*expr*, *ctx*)

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_EXISTS**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_FLOOR**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-floor

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_HOURS**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_IF**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-if

Parameters
expr (*Expr*) –

rdflib.plugins.sparql.operators.**Builtin_IRI**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-iri

Parameters

- **expr** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
URIRef

rdflib.plugins.sparql.operators.**Builtin_LANG**(*e*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-lang

Returns the language tag of ltrl, if it has one. It returns "" if ltrl has no language tag. Note that the RDF data model does not include literals with an empty language tag.

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_LANGMATCHES**(*e*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-langMatches

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_LCASE**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_MD5**(*expr*, *ctx*)

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_MINUTES**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_MONTH**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_NOW**(*e*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-now

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_RAND**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#idp2133952

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_REGEX**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-regex Invokes the XPath fn:matches function to match text against a regular expression pattern. The regular expression language is defined in XQuery 1.0 and XPath 2.0 Functions and Operators section 7.6.1 Regular Expression Syntax

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_REPLACE**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-substr

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_ROUND**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-round

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_SECONDS**(*e*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-seconds

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_SHA1**(*expr*, *ctx*)

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_SHA256**(*expr*, *ctx*)

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_SHA384**(*expr*, *ctx*)

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_SHA512**(*expr*, *ctx*)

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STR**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRAFTER**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strafter

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRBEFORE**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strbefore

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRDT**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strdt

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRENDS**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strends

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRLANG**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strlang

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRLEN**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRSTARTS**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strstarts

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_STRUUID**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strdt

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_SUBSTR**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-substr

Parameters
expr (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_TIMEZONE**(*e*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-timezone

Return type
Literal

Returns
the timezone part of arg as an xsd:dayTimeDuration.

Raises
an error if there is no timezone.

Parameters
e (*Expr*) –

rdflib.plugins.sparql.operators.**Builtin_TZ**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_UCASE**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_UUID**(*expr*, *ctx*)

http://www.w3.org/TR/sparql11-query/#func-strdt

Parameters
expr (*Expr*) –

Return type
URIRef

rdflib.plugins.sparql.operators.**Builtin_YEAR**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_isBLANK**(*expr*, *ctx*)

Parameters

- **expr** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_isIRI**(*expr*, *ctx*)

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_isLITERAL**(*expr*, *ctx*)

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_isNUMERIC**(*expr*, *ctx*)

Return type
Literal

rdflib.plugins.sparql.operators.**Builtin_sameTerm**(*e*, *ctx*)

Parameters
e (*Expr*) –

Return type
Literal

rdflib.plugins.sparql.operators.**ConditionalAndExpression**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (Union[*QueryContext*, *FrozenBindings*]) –

Return type
Literal

rdflib.plugins.sparql.operators.**ConditionalOrExpression**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (Union[*QueryContext*, *FrozenBindings*]) –

Return type
Literal

rdflib.plugins.sparql.operators.**EBV**(*rt:* Literal) → bool

rdflib.plugins.sparql.operators.**EBV**(*rt: Union[*Variable, IdentifiedNode, SPARQLError, Expr*]*) → NoReturn

rdflib.plugins.sparql.operators.**EBV**(*rt: Union[*Identifier, SPARQLError, Expr*]*) → Union[bool, NoReturn]

Effective Boolean Value (EBV)

- If the argument is a typed literal with a datatype of xsd:boolean, the EBV is the value of that argument.
- If the argument is a plain literal or a typed literal with a datatype of xsd:string, the EBV is false if the operand value has zero length; otherwise the EBV is true.
- If the argument is a numeric type or a typed literal with a datatype derived from a numeric type, the EBV is false if the operand value is NaN or is numerically equal to zero; otherwise the EBV is true.
- All other arguments, including unbound arguments, produce a type error.

Parameters
rt (Union[*Identifier*, *SPARQLError*, *Expr*]) –

Return type
bool

rdflib.plugins.sparql.operators.**Function**(*e*, *ctx*)

Custom functions and casts

Parameters

- **e** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Node

rdflib.plugins.sparql.operators.**MultiplicativeExpression**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (Union[*QueryContext*, *FrozenBindings*]) –

Return type
Literal

rdflib.plugins.sparql.operators.**RelationalExpression**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (Union[*QueryContext*, *FrozenBindings*]) –

Return type
Literal

rdflib.plugins.sparql.operators.**UnaryMinus**(*expr*, *ctx*)

Parameters

- **expr** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Literal

rdflib.plugins.sparql.operators.**UnaryNot**(*expr*, *ctx*)

Parameters

- **expr** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Literal

rdflib.plugins.sparql.operators.**UnaryPlus**(*expr*, *ctx*)

Parameters

- **expr** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Literal

rdflib.plugins.sparql.operators.**and_**(**args*)

Parameters
args (*Expr*) –

Return type
Expr

rdflib.plugins.sparql.operators.**calculateDuration**(*obj1*, *obj2*)
returns the duration Literal between two datetime

Parameters

- **obj1** (Union[date, datetime]) –
- **obj2** (Union[date, datetime]) –

Return type
Literal

rdflib.plugins.sparql.operators.**calculateFinalDateTime**(*obj1*, *dt1*, *obj2*, *dt2*, *operation*)
Calculates the final dateTime/date/time resultant after addition/ subtraction of duration/dayTimeDuration/yearMonthDuration

Parameters

- **obj1** (Union[date, datetime]) –
- **dt1** (*URIRef*) –
- **obj2** (Union[Duration, timedelta]) –

- **dt2** (*URIRef*) –
- **operation** (str) –

Return type
Literal

rdflib.plugins.sparql.operators.**custom_function**(*uri*, *override=False*, *raw=False*)

Decorator version of *register_custom_function()*.

Parameters

- **uri** (*URIRef*) –
- **override** (bool) –
- **raw** (bool) –

Return type
Callable[[Callable[[*Expr*, *FrozenBindings*], *Node*]], Callable[[*Expr*, *FrozenBindings*], *Node*]]

rdflib.plugins.sparql.operators.**date**(*e*)

Parameters
e (*Literal*) –

Return type
date

rdflib.plugins.sparql.operators.**dateTimeObjects**(*expr*)

return a dataTime/date/time/duration/dayTimeDuration/yearMonthDuration python objects from a literal

Parameters
expr (*Literal*) –

Return type
Any

rdflib.plugins.sparql.operators.**datetime**(*e*)

Parameters
e (*Literal*) –

Return type
datetime

rdflib.plugins.sparql.operators.**default_cast**(*e*, *ctx*)

Parameters

- **e** (*Expr*) –
- **ctx** (*FrozenBindings*) –

Return type
Literal

rdflib.plugins.sparql.operators.**isCompatibleDateTimeDatatype**(*obj1*, *dt1*, *obj2*, *dt2*)

Returns a boolean indicating if first object is compatible with operation(+/-) over second object.

Parameters

- **obj1** (Union[date, datetime]) –
- **dt1** (*URIRef*) –

- **obj2** (Union[Duration, timedelta]) –
- **dt2** (*URIRef*) –

Return type
bool

rdflib.plugins.sparql.operators.**literal**(*s*)

Parameters
s (*Literal*) –

Return type
Literal

rdflib.plugins.sparql.operators.**not_**(*arg*)

Return type
Expr

rdflib.plugins.sparql.operators.**numeric**(*expr*)

return a number from a literal http://www.w3.org/TR/xpath20/#promotion

or TypeError

Parameters
expr (*Literal*) –

Return type
Any

rdflib.plugins.sparql.operators.**register_custom_function**(*uri*, *func*, *override=False*, *raw=False*)

Register a custom SPARQL function.

By default, the function will be passed the RDF terms in the argument list. If raw is True, the function will be passed an Expression and a Context.

The function must return an RDF term, or raise a SparqlError.

Parameters

- **uri** (*URIRef*) –
- **func** (Callable[[*Expr*, *FrozenBindings*], *Node*]) –
- **override** (bool) –
- **raw** (bool) –

Return type
None

rdflib.plugins.sparql.operators.**simplify**(*expr*)

Parameters
expr (Any) –

Return type
Any

rdflib.plugins.sparql.operators.**string**(*s*)

Make sure the passed thing is a string literal i.e. plain literal, xsd:string literal or lang-tagged literal

Parameters
s (*Literal*) –

Return type
Literal

rdflib.plugins.sparql.operators.**unregister_custom_function**(*uri*, *func=None*)

The 'func' argument is included for compatibility with existing code. A previous implementation checked that the function associated with the given uri was actually 'func', but this is not necessary as the uri should uniquely identify the function.

Parameters

- **uri** (*URIRef*) –
- **func** (Optional[Callable[..., Any]]) –

Return type
None

rdflib.plugins.sparql.parser module

SPARQL 1.1 Parser

based on pyparsing

rdflib.plugins.sparql.parser.**expandBNodeTriples**(*terms*)

expand [?p ?o] syntax for implicit bnodes

Parameters
terms (ParseResults) –

Return type
List[Any]

rdflib.plugins.sparql.parser.**expandCollection**(*terms*)

expand (1 2 3) notation for collections

Parameters
terms (ParseResults) –

Return type
List[List[Any]]

rdflib.plugins.sparql.parser.**expandTriples**(*terms*)

Expand ; and , syntax for repeat predicates, subjects

Parameters
terms (ParseResults) –

Return type
List[Any]

rdflib.plugins.sparql.parser.**expandUnicodeEscapes**(*q*)

The syntax of the SPARQL Query Language is expressed over code points in Unicode [UNICODE]. The encoding is always UTF-8 [RFC3629]. Unicode code points may also be expressed using an uXXXX (U+0 to U+FFFF) or UXXXXXXXX syntax (for U+10000 onwards) where X is a hexadecimal digit [0-9A-F]

Parameters
q (str) –

Return type
str

rdflib.plugins.sparql.parser.**neg**(*literal*)

Parameters
literal (*Literal*) –

Return type
Literal

rdflib.plugins.sparql.parser.**parseQuery**(*q*)

Parameters
q (Union[str, bytes, TextIO, BinaryIO]) –

Return type
ParseResults

rdflib.plugins.sparql.parser.**parseUpdate**(*q*)

Parameters
q (Union[str, bytes, TextIO, BinaryIO]) –

Return type
CompValue

rdflib.plugins.sparql.parser.**setDataType**(*terms*)

Parameters
terms (Tuple[Any, Optional[str]]) –

Return type
Literal

rdflib.plugins.sparql.parser.**setLanguage**(*terms*)

Parameters
terms (Tuple[Any, Optional[str]]) –

Return type
Literal

rdflib.plugins.sparql.parserutils module

class rdflib.plugins.sparql.parserutils.**Comp**(*name*, *expr*)

Bases: TokenConverter

A pyparsing token for grouping together things with a label Any sub-tokens that are not Params will be ignored.

Returns CompValue / Expr objects - depending on whether evalFn is set.

Parameters
name (str) –

__abstractmethods__ = frozenset({})

__init__(*name*, *expr*)

Parameters
name (str) –

__module__ = 'rdflib.plugins.sparql.parserutils'

__slotnames__ = []

postParse(*instring*, *loc*, *tokenList*)

Parameters

- **instring** (str) –
- **loc** (int) –
- **tokenList** (ParseResults) –

Return type
Union[*Expr*, *CompValue*]

setEvalFn(*evalfn*)

Parameters
evalfn (Callable[[Any, Any], Any]) –

Return type
Comp

class rdflib.plugins.sparql.parserutils.**CompValue**(*name*, ***values*)

Bases: OrderedDict

The result of parsing a Comp Any included Params are available as Dict keys or as attributes

Parameters
name (str) –

__getattr__(*a*)

Parameters
a (str) –

Return type
Any

__getitem__(*a*)

x.__getitem__(y) <==> x[y]

__init__(*name*, ***values*)

Parameters
name (str) –

__module__ = 'rdflib.plugins.sparql.parserutils'

__repr__()

Return repr(self).

Return type
str

__str__()

Return str(self).

Return type
str

clone()

Return type
CompValue

get(*a*, *variables=False*, *errors=False*)

Return the value for key if key is in the dictionary, else default.

Parameters

- **variables** (bool) –
- **errors** (bool) –

class rdflib.plugins.sparql.parserutils.**Expr**(*name*, *evalfn=None*, ***values*)

Bases: *CompValue*

A CompValue that is evaluatable

Parameters

- **name** (str) –
- **evalfn** (Optional[Callable[[Any, Any], Any]]) –

__init__(*name*, *evalfn=None*, ***values*)

Parameters

- **name** (str) –
- **evalfn** (Optional[Callable[[Any, Any], Any]]) –

__module__ = 'rdflib.plugins.sparql.parserutils'

eval(*ctx={}*)

Parameters
ctx (Any) –

Return type
Union[*SPARQLError*, Any]

class rdflib.plugins.sparql.parserutils.**Param**(*name*, *expr*, *isList=False*)

Bases: TokenConverter

A pyparsing token for labelling a part of the parse-tree if isList is true repeat occurrences of ParamList have their values merged in a list

Parameters

- **name** (str) –
- **isList** (bool) –

__abstractmethods__ = frozenset({})

__init__(*name*, *expr*, *isList=False*)

Parameters

- **name** (str) –
- **isList** (bool) –

__module__ = 'rdflib.plugins.sparql.parserutils'

__slotnames__ = []

postParse2(*tokenList*)

Parameters
tokenList (Union[List[Any], ParseResults]) –

Return type
ParamValue

class rdflib.plugins.sparql.parserutils.**ParamList**(*name*, *expr*)

Bases: *Param*

A shortcut for a Param with isList=True

Parameters
name (str) –

__abstractmethods__ = frozenset({})

__init__(*name*, *expr*)

Parameters
name (str) –

__module__ = 'rdflib.plugins.sparql.parserutils'

__slotnames__ = []

failAction: Optional[ParseFailAction]

ignoreExprs: List['ParserElement']

parseAction: List[ParseAction]

suppress_warnings_: List[Diagnostics]

class rdflib.plugins.sparql.parserutils.**ParamValue**(*name*, *tokenList*, *isList*)

Bases: object

The result of parsing a Param This just keeps the name/value All cleverness is in the CompValue

Parameters

- **name** (str) –
- **tokenList** (Union[List[Any], ParseResults]) –
- **isList** (bool) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.parserutils', '__doc__': '\n The result of parsing a Param\n This just keeps the name/value\n All cleverness is in the CompValue\n ', '__init__': <function ParamValue.__init__>, '__str__': <function ParamValue.__str__>, '__dict__': <attribute '__dict__' of 'ParamValue' objects>, '__weakref__': <attribute '__weakref__' of 'ParamValue' objects>, '__annotations__': {}})

__init__(*name*, *tokenList*, *isList*)

Parameters

- **name** (str) –
- **tokenList** (Union[List[Any], ParseResults]) –
- **isList** (bool) –

__module__ = 'rdflib.plugins.sparql.parserutils'

__str__()

Return str(self).

Return type
str

__weakref__

list of weak references to the object (if defined)

rdflib.plugins.sparql.parserutils.**prettify_parsetree**(*t*, *indent=''*, *depth=0*)

Parameters

- **t** (ParseResults) –
- **indent** (str) –
- **depth** (int) –

Return type
str

rdflib.plugins.sparql.parserutils.**value**(*ctx*, *val*, *variables=False*, *errors=False*)

utility function for evaluating something...

Variables will be looked up in the context Normally, non-bound vars is an error, set variables=True to return unbound vars

Normally, an error raises the error, set errors=True to return error

Parameters

- **ctx** (*FrozenBindings*) –
- **val** (Any) –
- **variables** (bool) –
- **errors** (bool) –

Return type
Any

rdflib.plugins.sparql.processor module

Code for tying SPARQL Engine into RDFLib

These should be automatically registered with RDFLib

class rdflib.plugins.sparql.processor.**SPARQLProcessor**(*graph*)

Bases: *Processor*

__init__(*graph*)

__module__ = 'rdflib.plugins.sparql.processor'

query(*strOrQuery*, *initBindings={}*, *initNs={}*, *base=None*, *DEBUG=False*)

Evaluate a query with the given initial bindings, and initial namespaces. The given base is used to resolve relative URIs in the query and will be overridden by any BASE given in the query.

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in `SERVICE` directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **strOrQuery** (Union[str, *Query*]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –
- **DEBUG** (bool) –

Return type
Mapping[str, Any]

class rdflib.plugins.sparql.processor.**SPARQLResult**(*res*)

Bases: *Result*

Parameters
res (Mapping[str, Any]) –

__init__(*res*)

Parameters
res (Mapping[str, Any]) –

__module__ = 'rdflib.plugins.sparql.processor'

askAnswer: Optional[bool]

graph: Optional['Graph']

vars: Optional[List['Variable']]

variables contained in the result.

class rdflib.plugins.sparql.processor.**SPARQLUpdateProcessor**(*graph*)

Bases: *UpdateProcessor*

__init__(*graph*)

__module__ = 'rdflib.plugins.sparql.processor'

update(*strOrQuery*, *initBindings={}*, *initNs={}*)

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in `SERVICE` directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **strOrQuery** (Union[str, *Update*]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **initNs** (Mapping[str, Any]) –

Return type
None

rdflib.plugins.sparql.processor.**prepareQuery**(*queryString*, *initNs={}*, *base=None*)

Parse and translate a SPARQL Query

Parameters

- **queryString** (str) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –

Return type
Query

rdflib.plugins.sparql.processor.**prepareUpdate**(*updateString*, *initNs={}*, *base=None*)

Parse and translate a SPARQL Update

Parameters

- **updateString** (str) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –

Return type
Update

rdflib.plugins.sparql.processor.**processUpdate**(*graph*, *updateString*, *initBindings={}*, *initNs={}*, *base=None*)

Process a SPARQL Update Request returns Nothing on success or raises Exceptions on error

Parameters

- **graph** (*Graph*) –
- **updateString** (str) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –

Return type
None

rdflib.plugins.sparql.sparql module

exception rdflib.plugins.sparql.sparql.**AlreadyBound**

Bases: *SPARQLError*

Raised when trying to bind a variable that is already bound!

__init__()

__module__ = 'rdflib.plugins.sparql.sparql'

class rdflib.plugins.sparql.sparql.**Bindings**(*outer=None*, *d=[]*)

Bases: MutableMapping

A single level of a stack of variable-value bindings. Each dict keeps a reference to the dict below it, any failed lookup is propegated back

In python 3.3 this could be a collections.ChainMap

Parameters
outer (Optional[*Bindings*]) –

__abstractmethods__ = frozenset({})

__contains__(*key*)

Parameters
key (Any) –

Return type
bool

__delitem__(*key*)

Parameters
key (str) –

Return type
None

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.sparql', '__doc__': '\n\n A single level of a stack of variable-value bindings.\n Each dict keeps a reference to the dict below it,\n any failed lookup is propegated back\n\n In python 3.3 this could be a collections.ChainMap\n ', '__init__': <function Bindings.__init__>, '__getitem__': <function Bindings.__getitem__>, '__contains__': <function Bindings.__contains__>, '__setitem__': <function Bindings.__setitem__>, '__delitem__': <function Bindings.__delitem__>, '__len__': <function Bindings.__len__>, '__iter__': <function Bindings.__iter__>, '__str__': <function Bindings.__str__>, '__repr__': <function Bindings.__repr__>, '__dict__': <attribute '__dict__' of 'Bindings' objects>, '__weakref__': <attribute '__weakref__' of 'Bindings' objects>, '__abstractmethods__': frozenset(), '_abc_impl': <_abc._abc_data object>, '__annotations__': {'_d': 'Dict[str, str]'}})

__getitem__(*key*)

Parameters
key (str) –

Return type
str

__init__(*outer=None*, *d=[]*)

Parameters
outer (Optional[*Bindings*]) –

__iter__()

Return type
Generator[str, None, None]

__len__()

Return type
int

__module__ = 'rdflib.plugins.sparql.sparql'

__repr__()
Return repr(self).

Return type
str

__setitem__(*key*, *value*)

Parameters
- **key** (str) –
- **value** (Any) –

Return type
None

__str__()
Return str(self).

Return type
str

__weakref__
list of weak references to the object (if defined)

class rdflib.plugins.sparql.sparql.**FrozenBindings**(*ctx*, **args*, ***kwargs*)

Bases: *FrozenDict*

Parameters
ctx (*QueryContext*) –

__abstractmethods__ = frozenset({})

__getitem__(*key*)

Parameters
key (Union[*Identifier*, str]) –

Return type
Identifier

__init__(*ctx*, **args*, ***kwargs*)

Parameters
ctx (*QueryContext*) –

__module__ = 'rdflib.plugins.sparql.sparql'

property bnodes: Mapping[*Identifier*, *BNode*]

Return type
Mapping[*Identifier*, *BNode*]

forget(*before*, *_except=None*)
return a frozen dict only of bindings made in self since before

Parameters
- **before** (*QueryContext*) –
- **_except** (Optional[Container[*Variable*]]) –

Return type
FrozenBindings

merge(*other*)

Parameters
other (Mapping[*Identifier*, *Identifier*]) –

Return type
FrozenBindings

property now: datetime

Return type
datetime

project(*vars*)

Parameters
vars (Container[*Variable*]) –

Return type
FrozenBindings

property prologue: Optional[*Prologue*]

Return type
Optional[*Prologue*]

remember(*these*)

return a frozen dict only of bindings in these

Return type
FrozenBindings

class rdflib.plugins.sparql.sparql.**FrozenDict**(**args*, ***kwargs*)

Bases: Mapping

An immutable hashable dict

Taken from http://stackoverflow.com/a/2704866/81121

Parameters

- **args** (Any) –
- **kwargs** (Any) –

__abstractmethods__ = frozenset({})

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.sparql', '__doc__': '\n An immutable hashable dict\n\n Taken from http://stackoverflow.com/a/2704866/81121\n\n ', '__init__': <function FrozenDict.__init__>, '__iter__': <function FrozenDict.__iter__>, '__len__': <function FrozenDict.__len__>, '__getitem__': <function FrozenDict.__getitem__>, '__hash__': <function FrozenDict.__hash__>, 'project': <function FrozenDict.project>, 'disjointDomain': <function FrozenDict.disjointDomain>, 'compatible': <function FrozenDict.compatible>, 'merge': <function FrozenDict.merge>, '__str__': <function FrozenDict.__str__>, '__repr__': <function FrozenDict.__repr__>, '__dict__': <attribute '__dict__' of 'FrozenDict' objects>, '__weakref__': <attribute '__weakref__' of 'FrozenDict' objects>, '__abstractmethods__': frozenset(), '_abc_impl': <_abc._abc_data object>, '__annotations__': {'_d': 'Dict[Identifier, Identifier]', '_hash': 'Optional[int]'}})

__getitem__(*key*)

Parameters
key (*Identifier*) –

Return type
Identifier

__hash__()

Return hash(self).

Return type
int

__init__(**args*, ***kwargs*)

Parameters

- **args** (Any) –
- **kwargs** (Any) –

__iter__()

__len__()

Return type
int

__module__ = 'rdflib.plugins.sparql.sparql'

__repr__()
Return repr(self).

Return type
str

__str__()
Return str(self).

Return type
str

__weakref__
list of weak references to the object (if defined)

compatible(*other*)

Parameters
other (Mapping[*Identifier*, *Identifier*]) –

Return type
bool

disjointDomain(*other*)

Parameters
other (Mapping[*Identifier*, *Identifier*]) –

Return type
bool

merge(*other*)

Parameters
other (Mapping[*Identifier*, *Identifier*]) –

Return type
FrozenDict

project(*vars*)

Parameters
vars (Container[*Variable*]) –

Return type
FrozenDict

exception rdflib.plugins.sparql.sparql.**NotBoundError**(*msg=None*)
Bases: *SPARQLError*

Parameters
msg (Optional[str]) –

__init__(*msg=None*)

Parameters
msg (Optional[str]) –

__module__ = 'rdflib.plugins.sparql.sparql'

class rdflib.plugins.sparql.sparql.**Prologue**

Bases: object

A class for holding prefixing bindings and base URI information

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.sparql', '__doc__': '\n A class for holding prefixing bindings and base URI information\n ', '__init__': <function Prologue.__init__>, 'resolvePName': <function Prologue.resolvePName>, 'bind': <function Prologue.bind>, 'absolutize': <function Prologue.absolutize>, '__dict__': <attribute '__dict__' of 'Prologue' objects>, '__weakref__': <attribute '__weakref__' of 'Prologue' objects>, '__annotations__': {'base': 'Optional[str]'}})

__init__()

__module__ = 'rdflib.plugins.sparql.sparql'

__weakref__

list of weak references to the object (if defined)

absolutize(*iri*)

Apply BASE / PREFIXes to URIs (and to datatypes in Literals)

TODO: Move resolving URIs to pre-processing

Parameters
iri (Union[*CompValue*, str, None]) –

Return type
Union[*CompValue*, str, None]

bind(*prefix*, *uri*)

Parameters

- **prefix** (Optional[str]) –
- **uri** (Any) –

Return type
None

resolvePName(*prefix*, *localname*)

Parameters

- **prefix** (Optional[str]) –
- **localname** (Optional[str]) –

Return type
URIRef

class rdflib.plugins.sparql.sparql.**Query**(*prologue*, *algebra*)

Bases: object

A parsed and translated query

Parameters

- **prologue** (*Prologue*) –
- **algebra** (*CompValue*) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.sparql', '__doc__': '\n A parsed and translated query\n ', '__init__': <function Query.__init__>, '__dict__': <attribute '__dict__' of 'Query' objects>, '__weakref__': <attribute '__weakref__' of 'Query' objects>, '__annotations__': {'_original_args': 'Tuple[str, Mapping[str, str], Optional[str]]'}})

__init__(*prologue*, *algebra*)

Parameters

- **prologue** (*Prologue*) –
- **algebra** (*CompValue*) –

__module__ = 'rdflib.plugins.sparql.sparql'

__weakref__

list of weak references to the object (if defined)

class rdflib.plugins.sparql.sparql.**QueryContext**(*graph=None*, *bindings=None*, *initBindings=None*)

Bases: object

Query context - passed along when evaluating the query

Parameters

- **graph** (Optional[*Graph*]) –
- **bindings** (Union[*Bindings*, *FrozenBindings*, List[Any], None]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.sparql', '__doc__': '\n Query context - passed along when evaluating the query\n ', '__init__': <function QueryContext.__init__>, 'now': <property object>, 'clone': <function QueryContext.clone>, 'dataset': <property object>, 'load': <function QueryContext.load>, '__getitem__': <function QueryContext.__getitem__>, 'get': <function QueryContext.get>, 'solution': <function QueryContext.solution>, '__setitem__': <function QueryContext.__setitem__>, 'pushGraph': <function QueryContext.pushGraph>, 'push': <function QueryContext.push>, 'clean': <function QueryContext.clean>, 'thaw': <function QueryContext.thaw>, '__dict__': <attribute '__dict__' of 'QueryContext' objects>, '__weakref__': <attribute '__weakref__' of 'QueryContext' objects>, '__annotations__': {'graph': 'Optional[Graph]', '_dataset': 'Optional[ConjunctiveGraph]', 'prologue': 'Optional[Prologue]', '_now': 'Optional[datetime.datetime]', 'bnodes': 't.MutableMapping[Identifier, BNode]'}})

__getitem__(*key*)

Parameters

key (Union[str, *Path*]) –

Return type
Union[str, *Path*, None]

__init__(*graph=None*, *bindings=None*, *initBindings=None*)

Parameters

- **graph** (Optional[*Graph*]) –
- **bindings** (Union[*Bindings*, *FrozenBindings*, List[Any], None]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –

__module__ = 'rdflib.plugins.sparql.sparql'

__setitem__(*key*, *value*)

Parameters

- **key** (str) –
- **value** (str) –

Return type
None

__weakref__
list of weak references to the object (if defined)

clean()

Return type
QueryContext

clone(*bindings=None*)

Parameters
bindings (Union[*Bindings*, *FrozenBindings*, List[Any], None]) –

Return type
QueryContext

property dataset: *ConjunctiveGraph*
"current dataset

Return type
ConjunctiveGraph

get(*key*, *default=None*)

Parameters

- **key** (str) –
- **default** (Optional[Any]) –

Return type
Any

load(*source*, *default=False*, ***kwargs*)

Parameters

- **source** (*URIRef*) –
- **default** (bool) –

- **kwargs** (Any) –

Return type
None

property now: datetime

Return type
datetime

push()

Return type
QueryContext

pushGraph(*graph*)

Parameters
graph (Optional[*Graph*]) –

Return type
QueryContext

solution(*vars=None*)

Return a static copy of the current variable bindings as dict

Parameters
vars (Optional[Iterable[*Variable*]]) –

Return type
FrozenBindings

thaw(*frozenbindings*)

Create a new read/write query context from the given solution

Parameters
frozenbindings (*FrozenBindings*) –

Return type
QueryContext

exception rdflib.plugins.sparql.sparql.**SPARQLError**(*msg=None*)

Bases: Exception

Parameters
msg (Optional[str]) –

__init__(*msg=None*)

Parameters
msg (Optional[str]) –

__module__ = 'rdflib.plugins.sparql.sparql'

__weakref__

list of weak references to the object (if defined)

exception rdflib.plugins.sparql.sparql.**SPARQLTypeError**(*msg*)

Bases: *SPARQLError*

Parameters
msg (Optional[str]) –

__init__(*msg*)

Parameters
msg (Optional[str]) –

__module__ = 'rdflib.plugins.sparql.sparql'

class rdflib.plugins.sparql.sparql.**Update**(*prologue*, *algebra*)

Bases: object

A parsed and translated update

Parameters

- **prologue** (*Prologue*) –
- **algebra** (List[*CompValue*]) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.sparql.sparql', '__doc__': '\n A parsed and translated update\n ', '__init__': <function Update.__init__>, '__dict__': <attribute '__dict__' of 'Update' objects>, '__weakref__': <attribute '__weakref__' of 'Update' objects>, '__annotations__': {'_original_args': 'Tuple[str, Mapping[str, str], Optional[str]]'}})

__init__(*prologue*, *algebra*)

Parameters

- **prologue** (*Prologue*) –
- **algebra** (List[*CompValue*]) –

__module__ = 'rdflib.plugins.sparql.sparql'

__weakref__
list of weak references to the object (if defined)

rdflib.plugins.sparql.update module

Code for carrying out Update Operations

rdflib.plugins.sparql.update.**evalAdd**(*ctx*, *u*)

add all triples from src to dst

http://www.w3.org/TR/sparql11-update/#add

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalClear**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#clear

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalCopy**(*ctx*, *u*)

remove all triples from dst add all triples from src to dst

http://www.w3.org/TR/sparql11-update/#copy

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalCreate**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#create

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalDeleteData**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#deleteData

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalDeleteWhere**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#deleteWhere

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalDrop**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#drop

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalInsertData**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#insertData

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalLoad**(*ctx*, *u*)

http://www.w3.org/TR/sparql11-update/#load

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalModify**(*ctx*, *u*)

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalMove**(*ctx*, *u*)

remove all triples from dst add all triples from src to dst remove all triples from src

http://www.w3.org/TR/sparql11-update/#move

Parameters

- **ctx** (*QueryContext*) –
- **u** (*CompValue*) –

Return type
None

rdflib.plugins.sparql.update.**evalUpdate**(*graph*, *update*, *initBindings={}*)

http://www.w3.org/TR/sparql11-update/#updateLanguage

'A request is a sequence of operations [...] Implementations MUST ensure that operations of a single request are executed in a fashion that guarantees the same effects as executing them in lexical order.

Operations all result either in success or failure.

If multiple operations are present in a single request, then a result of failure from any operation MUST abort the sequence of operations, causing the subsequent operations to be ignored.'

This will return None on success and raise Exceptions on error

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in `SERVICE` directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **graph** (*Graph*) –
- **update** (*Update*) –
- **initBindings** (Mapping[str, *Identifier*]) –

Return type
None

Module contents

SPARQL implementation for RDFLib

New in version 4.0.

`rdflib.plugins.sparql.`**CUSTOM_EVALS = {}**

Custom evaluation functions

These must be functions taking (ctx, part) and raise NotImplementedError if they cannot handle a certain part

`rdflib.plugins.sparql.`**prepareQuery**(*queryString*, *initNs={}*, *base=None*)

Parse and translate a SPARQL Query

Parameters

- **queryString** (str) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –

Return type
Query

`rdflib.plugins.sparql.`**prepareUpdate**(*updateString*, *initNs={}*, *base=None*)

Parse and translate a SPARQL Update

Parameters

- **updateString** (str) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –

Return type
Update

`rdflib.plugins.sparql.`**processUpdate**(*graph*, *updateString*, *initBindings={}*, *initNs={}*, *base=None*)

Process a SPARQL Update Request returns Nothing on success or raises Exceptions on error

Parameters

- **graph** (*Graph*) –
- **updateString** (str) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **initNs** (Mapping[str, Any]) –
- **base** (Optional[str]) –

Return type
None

rdflib.plugins.stores package

Submodules

rdflib.plugins.stores.auditable module

This wrapper intercepts calls through the store interface and implements thread-safe logging of destructive operations (adds / removes) in reverse. This is persisted on the store instance and the reverse operations are executed In order to return the store to the state it was when the transaction began Since the reverse operations are persisted on the store, the store itself acts as a transaction.

Calls to commit or rollback, flush the list of reverse operations This provides thread-safe atomicity and isolation (assuming concurrent operations occur with different store instances), but no durability (transactions are persisted in memory and wont be available to reverse operations after the system fails): A and I out of ACID.

class rdflib.plugins.stores.auditable.**AuditableStore**(*store*)

Bases: *Store*

Parameters
store (*Store*) –

__init__(*store*)

identifier: URIRef of the Store. Defaults to CWD configuration: string containing information open can use to connect to datastore.

Parameters
store (*Store*) –

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters
context (Optional[*Graph*]) – a graph instance to query or None

__module__ = 'rdflib.plugins.stores.auditable'

add(*triple*, *context*, *quoted=False*)

Adds the given statement to a specific context or to the model. The quoted argument is interpreted by formula-aware stores to indicate this statement is quoted/hypothetical It should be an error to not specify a context and have the quoted argument be True. It should also be an error for the quoted argument to be True when the store is not formula-aware.

Parameters

- **triple** (Tuple[*Node*, *Node*, *Node*]) –

- **context** (*Graph*) –
- **quoted** (bool) –

Return type
None

bind(*prefix*, *namespace*, *override=True*)

Parameters

- **override** (bool) – rebind, even if the given namespace is already bound to another prefix.
- **prefix** (str) –
- **namespace** (*URIRef*) –

Return type
None

close(*commit_pending_transaction=False*)

This closes the database connection. The commit_pending_transaction parameter specifies whether to commit all pending transactions before closing (if the store is transactional).

Parameters
commit_pending_transaction (bool) –

Return type
None

commit()

Return type
None

contexts(*triple=None*)

Generator over all contexts in the graph. If triple is specified, a generator over all contexts the triple is in.

if store is graph_aware, may also return empty contexts

Return type
Generator[*Graph*, None, None]

Returns
a generator over Nodes

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

destroy(*configuration*)

This destroys the instance of the store identified by the configuration string.

Parameters
configuration (str) –

Return type
None

namespace(*prefix*)

Parameters
prefix (str) –

Return type
Optional[*URIRef*]

namespaces()

Return type
Iterator[Tuple[str, *URIRef*]]

open(*configuration*, *create=True*)

Opens the store specified by the configuration string. If create is True a store will be created if it does not already exist. If create is False and a store does not already exist an exception is raised. An exception is also raised if a store exists, but there is insufficient permissions to open the store. This should return one of: VALID_STORE, CORRUPTED_STORE, or NO_STORE

Parameters

- **configuration** (str) –
- **create** (bool) –

Return type
Optional[int]

prefix(*namespace*)

Parameters
namespace (*URIRef*) –

Return type
Optional[str]

query(**args*, ***kw*)

If stores provide their own SPARQL implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **args** (Any) –
- **kw** (Any) –

Return type
Result

remove(*spo*, *context=None*)

Remove the set of triples matching the pattern from the store

Parameters

- **spo** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type
None

rollback()

Return type
None

triples(*triple*, *context=None*)

A generator over all the triples matching the pattern. Pattern can include any objects for used for comparing against nodes in the store, for example, REGEXTerm, URIRef, Literal, BNode, Variable, Graph, QuotedGraph, Date? DateRange?

Parameters

- **context** (Optional[*Graph*]) – A conjunctive query can be indicated by either providing a value of None, or a specific context can be queries by passing a Graph instance (if store is context aware).
- **triple** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type

Iterator[Tuple[Tuple[*Node*, *Node*, *Node*], Iterator[Optional[*Graph*]]]]

rdflib.plugins.stores.berkeleydb module

class rdflib.plugins.stores.berkeleydb.**BerkeleyDB**(*configuration=None*, *identifier=None*)

Bases: *Store*

A store that allows for on-disk persistent using BerkeleyDB, a fast key/value DB.

This store implementation used to be known, previous to rdflib 6.0.0 as 'Sleepycat' due to that being the then name of the Python wrapper for BerkeleyDB.

This store allows for quads as well as triples. See examples of use in both the *examples.berkeleydb_example* and `test/test_store/test_store_berkeleydb.py` files.

NOTE on installation:

To use this store, you must have BerkeleyDB installed on your system separately to Python (`brew install berkeley-db` on a Mac) and also have the BerkeleyDB Python wrapper installed (`pip install berkeleydb`). You may need to install BerkeleyDB Python wrapper like this: `YES_I_HAVE_THE_RIGHT_TO_USE_THIS_BERKELEY_DB_VERSION=1 pip install berkeleydb`

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__annotations__ = {'db_env': 'db.DBEnv'}

__init__(*configuration=None*, *identifier=None*)

identifier: URIRef of the Store. Defaults to CWD configuration: string containing information open can use to connect to datastore.

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters

context (Optional[*Graph*]) – a graph instance to query or None

Return type
int

__module__ = 'rdflib.plugins.stores.berkeleydb'

add(*triple*, *context*, *quoted=False*, *txn=None*)

Add a triple to the store of triples.

Parameters

- **triple** (Tuple[*Node*, *Node*, *Node*]) –
- **context** (*Graph*) –
- **quoted** (bool) –
- **txn** (Optional[Any]) –

Return type
None

add_graph(*graph*)

Add a graph to the store, no effect if the graph already exists. :type graph: *Graph* :param graph: a Graph instance

Return type
None

bind(*prefix*, *namespace*, *override=True*)

Parameters

- **override** (bool) – rebind, even if the given namespace is already bound to another prefix.
- **prefix** (str) –
- **namespace** (*URIRef*) –

Return type
None

close(*commit_pending_transaction=False*)

This closes the database connection. The commit_pending_transaction parameter specifies whether to commit all pending transactions before closing (if the store is transactional).

Parameters
commit_pending_transaction (bool) –

Return type
None

context_aware: bool = True

contexts(*triple=None*)

Generator over all contexts in the graph. If triple is specified, a generator over all contexts the triple is in.

if store is graph_aware, may also return empty contexts

Return type
Generator[*Graph*, None, None]

Returns
a generator over Nodes

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

db_env: db.DBEnv = None

formula_aware: bool = True

graph_aware: bool = True

property identifier: Optional[*Identifier*]

Return type
Optional[*Identifier*]

is_open()

Return type
bool

namespace(*prefix*)

Parameters
prefix (str) –

Return type
Optional[*URIRef*]

namespaces()

Return type
Generator[Tuple[str, *URIRef*], None, None]

open(*path*, *create=True*)

Opens the store specified by the configuration string. If create is True a store will be created if it does not already exist. If create is False and a store does not already exist an exception is raised. An exception is also raised if a store exists, but there is insufficient permissions to open the store. This should return one of: VALID_STORE, CORRUPTED_STORE, or NO_STORE

Parameters

- **path** (str) –
- **create** (bool) –

Return type
Optional[int]

prefix(*namespace*)

Parameters
namespace (*URIRef*) –

Return type
Optional[str]

remove(*spo*, *context*, *txn=None*)

Remove the set of triples matching the pattern from the store

Parameters

- **spo** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

- **txn** (Optional[Any]) –

Return type
None

remove_graph(*graph*)

Remove a graph from the store, this should also remove all triples in the graph

Parameters

- **graphid** – a Graph instance
- **graph** (*Graph*) –

sync()

Return type
None

transaction_aware: bool = False

triples(*spo*, *context=None*, *txn=None*)

A generator over all the triples matching

Parameters

- **spo** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –
- **txn** (Optional[Any]) –

Return type
Generator[Tuple[Tuple[*Node*, *Node*, *Node*], Generator[Optional[*Graph*], None, None]], None, None]

rdflib.plugins.stores.concurrent module

class rdflib.plugins.stores.concurrent.**ConcurrentStore**(*store*)

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.stores.concurrent', '__init__': <function ConcurrentStore.__init__>, 'add': <function ConcurrentStore.add>, 'remove': <function ConcurrentStore.remove>, 'triples': <function ConcurrentStore.triples>, '__len__': <function ConcurrentStore.__len__>, '_ConcurrentStore__begin_read': <function ConcurrentStore.__begin_read>, '_ConcurrentStore__end_read': <function ConcurrentStore.__end_read>, '__dict__': <attribute '__dict__' of 'ConcurrentStore' objects>, '__weakref__': <attribute '__weakref__' of 'ConcurrentStore' objects>, '__doc__': None, '__annotations__': {}})

__init__(*store*)

__len__()

__module__ = 'rdflib.plugins.stores.concurrent'

__weakref__

list of weak references to the object (if defined)

add(*triple*)

remove(*triple*)

triples(*triple*)

class rdflib.plugins.stores.concurrent.**ResponsibleGenerator**(*gen*, *cleanup*)

Bases: object

A generator that will help clean up when it is done being used.

__del__()

__init__(*gen*, *cleanup*)

__iter__()

__module__ = 'rdflib.plugins.stores.concurrent'

__next__()

__slots__ = ['cleanup', 'gen']

cleanup

gen

rdflib.plugins.stores.memory module

class rdflib.plugins.stores.memory.**Memory**(*configuration=None*, *identifier=None*)

Bases: *Store*

An in memory implementation of a triple store.

Same as SimpleMemory above, but is Context-aware, Graph-aware, and Formula-aware Authors: Ashley Sommer

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__init__(*configuration=None*, *identifier=None*)

identifier: URIRef of the Store. Defaults to CWD configuration: string containing information open can use to connect to datastore.

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters

context (Optional[*Graph*]) – a graph instance to query or None

Return type

int

__module__ = 'rdflib.plugins.stores.memory'

add(*triple*, *context*, *quoted=False*)

Add a triple to the store of triples.

Parameters

- **triple** (Tuple[*Node*, *Node*, *Node*]) –
- **context** (*Graph*) –
- **quoted** (bool) –

Return type
None

add_graph(*graph*)

Add a graph to the store, no effect if the graph already exists. :type graph: *Graph* :param graph: a Graph instance

Return type
None

bind(*prefix*, *namespace*, *override=True*)

Parameters

- **override** (bool) – rebind, even if the given namespace is already bound to another prefix.
- **prefix** (str) –
- **namespace** (*URIRef*) –

Return type
None

context_aware: bool = True

contexts(*triple=None*)

Generator over all contexts in the graph. If triple is specified, a generator over all contexts the triple is in.

if store is graph_aware, may also return empty contexts

Return type
Generator[*Graph*, None, None]

Returns
a generator over Nodes

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

formula_aware: bool = True

graph_aware: bool = True

namespace(*prefix*)

Parameters
prefix (str) –

Return type
Optional[*URIRef*]

namespaces()

Return type
Iterator[Tuple[str, *URIRef*]]

prefix(*namespace*)

Parameters
namespace (*URIRef*) –

Return type
Optional[str]

query(*query*, *initNs*, *initBindings*, *queryGraph*, ***kwargs*)

If stores provide their own SPARQL implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **query** (Union[*Query*, str]) –
- **initNs** (Mapping[str, Any]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **queryGraph** (str) –

Return type
Result

remove(*triple_pattern*, *context=None*)

Remove the set of triples matching the pattern from the store

Parameters

- **triple_pattern** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type
None

remove_graph(*graph*)

Remove a graph from the store, this should also remove all triples in the graph

Parameters

- **graphid** – a Graph instance
- **graph** (*Graph*) –

Return type
None

triples(*triple_pattern*, *context=None*)

A generator over all the triples matching

Parameters

- **triple_pattern** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Tuple[Tuple[*Node*, *Node*, *Node*], Generator[Optional[*Graph*], None, None]], None, None]

update(*update*, *initNs*, *initBindings*, *queryGraph*, ***kwargs*)

If stores provide their own (SPARQL) Update implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **update** (Union[*Update*, Any]) –
- **initNs** (Mapping[str, Any]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **queryGraph** (str) –

Return type
None

class rdflib.plugins.stores.memory.**SimpleMemory**(*configuration=None*, *identifier=None*)

Bases: *Store*

A fast naive in memory implementation of a triple store.

This triple store uses nested dictionaries to store triples. Each triple is stored in two such indices as follows spo[s][p][o] = 1 and pos[p][o][s] = 1.

Authors: Michel Pelletier, Daniel Krech, Stefan Niederhauser

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__init__(*configuration=None*, *identifier=None*)

identifier: URIRef of the Store. Defaults to CWD configuration: string containing information open can use to connect to datastore.

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters
context (Optional[*Graph*]) – a graph instance to query or None

Return type
int

__module__ = 'rdflib.plugins.stores.memory'

add(*triple*, *context*, *quoted=False*)

Add a triple to the store of triples.

Parameters

- **triple** (Tuple[*Node*, *Node*, *Node*]) –
- **context** (*Graph*) –
- **quoted** (bool) –

Return type
None

bind(*prefix*, *namespace*, *override=True*)

Parameters

- **override** (bool) – rebind, even if the given namespace is already bound to another prefix.
- **prefix** (str) –
- **namespace** (*URIRef*) –

Return type
None

namespace(*prefix*)

Parameters
prefix (str) –

Return type
Optional[*URIRef*]

namespaces()

Return type
Iterator[Tuple[str, *URIRef*]]

prefix(*namespace*)

Parameters
namespace (*URIRef*) –

Return type
Optional[str]

query(*query*, *initNs*, *initBindings*, *queryGraph*, ***kwargs*)

If stores provide their own SPARQL implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **query** (Union[*Query*, str]) –
- **initNs** (Mapping[str, Any]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **queryGraph** (str) –

- **kwargs** (Any) –

Return type
Result

remove(*triple_pattern*, *context=None*)

Remove the set of triples matching the pattern from the store

Parameters

- **triple_pattern** (Tuple[Optional[Node], Optional[Node], Optional[Node]]) –
- **context** (Optional[Graph]) –

Return type
None

triples(*triple_pattern*, *context=None*)

A generator over all the triples matching

Parameters

- **triple_pattern** (Tuple[Optional[Node], Optional[Node], Optional[Node]]) –
- **context** (Optional[Graph]) –

Return type
Iterator[Tuple[Tuple[Node, Node, Node], Iterator[Optional[Graph]]]]

update(*update*, *initNs*, *initBindings*, *queryGraph*, ***kwargs*)

If stores provide their own (SPARQL) Update implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **update** (Union[Update, str]) –
- **initNs** (Mapping[str, Any]) –
- **initBindings** (Mapping[str, Identifier]) –
- **queryGraph** (str) –
- **kwargs** (Any) –

Return type
None

rdflib.plugins.stores.regexmatching module

This wrapper intercepts calls through the store interface which make use of the REGEXTerm class to represent matches by REGEX instead of literal comparison.

Implemented for stores that don't support this and essentially provides the support by replacing the REGEXTerms by wildcards (None) and matching against the results from the store it's wrapping.

class rdflib.plugins.stores.regexmatching.**REGEXMatching**(*storage*)

Bases: Store

__init__(*storage*)

identifier: URIRef of the Store. Defaults to CWD configuration: string containing information open can use to connect to datastore.

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters
context – a graph instance to query or None

__module__ = 'rdflib.plugins.stores.regexmatching'

add(*triple*, *context*, *quoted=False*)

Adds the given statement to a specific context or to the model. The quoted argument is interpreted by formula-aware stores to indicate this statement is quoted/hypothetical It should be an error to not specify a context and have the quoted argument be True. It should also be an error for the quoted argument to be True when the store is not formula-aware.

bind(*prefix*, *namespace*, *override=True*)

Parameters
override – rebind, even if the given namespace is already bound to another prefix.

close(*commit_pending_transaction=False*)

This closes the database connection. The commit_pending_transaction parameter specifies whether to commit all pending transactions before closing (if the store is transactional).

commit()

contexts(*triple=None*)

Generator over all contexts in the graph. If triple is specified, a generator over all contexts the triple is in.

if store is graph_aware, may also return empty contexts

Returns
a generator over Nodes

destroy(*configuration*)

This destroys the instance of the store identified by the configuration string.

namespace(*prefix*)

namespaces()

open(*configuration*, *create=True*)

Opens the store specified by the configuration string. If create is True a store will be created if it does not already exist. If create is False and a store does not already exist an exception is raised. An exception is also raised if a store exists, but there is insufficient permissions to open the store. This should return one of: VALID_STORE, CORRUPTED_STORE, or NO_STORE

prefix(*namespace*)

remove(*triple*, *context=None*)

Remove the set of triples matching the pattern from the store

remove_context(*identifier*)

rollback()

triples(*triple*, *context=None*)

A generator over all the triples matching the pattern. Pattern can include any objects for used for comparing against nodes in the store, for example, REGEXTerm, URIRef, Literal, BNode, Variable, Graph, QuotedGraph, Date? DateRange?

Parameters

context – A conjunctive query can be indicated by either providing a value of None, or a specific context can be queries by passing a Graph instance (if store is context aware).

class rdflib.plugins.stores.regexmatching.**REGEXTerm**(*expr*)

Bases: str

REGEXTerm can be used in any term slot and is interpreted as a request to perform a REGEX match (not a string comparison) using the value (pre-compiled) for checking rdf:type matches

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.stores.regexmatching', '__doc__': '\n REGEXTerm can be used in any term slot and is interpreted as a request to\n perform a REGEX match (not a string comparison) using the value\n (pre-compiled) for checking rdf:type matches\n ', '__init__': <function REGEXTerm.__init__>, '__reduce__': <function REGEXTerm.__reduce__>, '__dict__': <attribute '__dict__' of 'REGEXTerm' objects>, '__weakref__': <attribute '__weakref__' of 'REGEXTerm' objects>, '__annotations__': {}})

__init__(*expr*)

__module__ = 'rdflib.plugins.stores.regexmatching'

__reduce__()

Helper for pickle.

__weakref__

list of weak references to the object (if defined)

rdflib.plugins.stores.regexmatching.**regexCompareQuad**(*quad*, *regexQuad*)

rdflib.plugins.stores.sparqlconnector module

class rdflib.plugins.stores.sparqlconnector.**SPARQLConnector**(*query_endpoint=None*, *update_endpoint=None*, *returnFormat='xml'*, *method='GET'*, *auth=None*, ***kwargs*)

Bases: object

this class deals with nitty gritty details of talking to a SPARQL server

Parameters

- **query_endpoint** (Optional[str]) –
- **update_endpoint** (Optional[str]) –
- **returnFormat** (str) –
- **method** (Literal['GET', 'POST', 'POST_FORM']) –
- **auth** (Optional[Tuple[str, str]]) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugins.stores.sparqlconnector', '__doc__': '\n this class deals with nitty gritty details of talking to a SPARQL server\n ', '__init__': <function SPARQLConnector.__init__>, 'method': <property object>, 'query': <function SPARQLConnector.query>, 'update': <function SPARQLConnector.update>, '__dict__': <attribute '__dict__' of 'SPARQLConnector' objects>, '__weakref__': <attribute '__weakref__' of 'SPARQLConnector' objects>, '__annotations__': {'_method': 'str'}})

__init__(*query_endpoint=None*, *update_endpoint=None*, *returnFormat='xml'*, *method='GET'*, *auth=None*, ***kwargs*)

auth, if present, must be a tuple of (username, password) used for Basic Authentication

Any additional keyword arguments will be passed to to the request, and can be used to setup timesouts etc.

Parameters

- **query_endpoint** (Optional[str]) –
- **update_endpoint** (Optional[str]) –
- **returnFormat** (str) –
- **method** (Literal['GET', 'POST', 'POST_FORM']) –
- **auth** (Optional[Tuple[str, str]]) –

__module__ = 'rdflib.plugins.stores.sparqlconnector'

__weakref__

list of weak references to the object (if defined)

property method: str

Return type
str

query(*query*, *default_graph=None*, *named_graph=None*)

Parameters

- **query** (str) –
- **default_graph** (Optional[str]) –
- **named_graph** (Optional[str]) –

Return type
Result

update(*query*, *default_graph=None*, *named_graph=None*)

Parameters

- **query** (str) –
- **default_graph** (Optional[str]) –
- **named_graph** (Optional[str]) –

Return type
None

exception rdflib.plugins.stores.sparqlconnector.SPARQLConnectorException

Bases: Exception

__module__ = 'rdflib.plugins.stores.sparqlconnector'

__weakref__

list of weak references to the object (if defined)

rdflib.plugins.stores.sparqlstore module

This is an RDFLib store around Ivan Herman et al.'s SPARQL service wrapper. This was first done in layer-cake, and then ported to RDFLib

class rdflib.plugins.stores.sparqlstore.**SPARQLStore**(*query_endpoint=None*, *sparql11=True*, *context_aware=True*, *node_to_sparql=<function _node_to_sparql>*, *returnFormat='xml'*, *auth=None*, ***sparqlconnector_kwargs*)

Bases: *SPARQLConnector*, *Store*

An RDFLib store around a SPARQL endpoint

This is context-aware and should work as expected when a context is specified.

For ConjunctiveGraphs, reading is done from the "default graph". Exactly what this means depends on your endpoint, because SPARQL does not offer a simple way to query the union of all graphs as it would be expected for a ConjuntiveGraph. This is why we recommend using Dataset instead, which is motivated by the SPARQL 1.1.

Fuseki/TDB has a flag for specifying that the default graph is the union of all graphs (`tdb:unionDefaultGraph` in the Fuseki config).

> **Warning:** By default the SPARQL Store does not support blank-nodes!
>
> As blank-nodes act as variables in SPARQL queries, there is no way to query for a particular blank node without using non-standard SPARQL extensions.
>
> See http://www.w3.org/TR/sparql11-query/#BGPsparqlBNodes

You can make use of such extensions through the `node_to_sparql` argument. For example if you want to transform BNode('0001') into "<bnode:b0001>", you can use a function like this:

```
>>> def my_bnode_ext(node):
...     if isinstance(node, BNode):
...         return '<bnode:b%s>' % node
...     return _node_to_sparql(node)
>>> store = SPARQLStore('http://dbpedia.org/sparql',
...                     node_to_sparql=my_bnode_ext)
```

You can request a particular result serialization with the `returnFormat` parameter. This is a string that must have a matching plugin registered. Built in is support for `xml`, `json`, `csv`, `tsv` and `application/rdf+xml`.

The underlying SPARQLConnector uses the urllib library. Any extra kwargs passed to the SPARQLStore connector are passed to urllib when doing HTTP calls. I.e. you have full control of cookies/auth/headers.

Form example:

```
>>> store = SPARQLStore('...my endpoint ...', auth=('user','pass'))
```

will use HTTP basic auth.

Parameters

- **query_endpoint** (Optional[str]) –
- **sparql11** (bool) –
- **context_aware** (bool) –
- **node_to_sparql** (Callable[[*Node*], str]) –
- **returnFormat** (str) –
- **auth** (Optional[Tuple[str, str]]) –

__init__(*query_endpoint=None*, *sparql11=True*, *context_aware=True*, *node_to_sparql=<function _node_to_sparql>*, *returnFormat='xml'*, *auth=None*, ***sparqlconnector_kwargs*)

auth, if present, must be a tuple of (username, password) used for Basic Authentication

Any additional keyword arguments will be passed to to the request, and can be used to setup timesouts etc.

Parameters

- **query_endpoint** (Optional[str]) –
- **sparql11** (bool) –
- **context_aware** (bool) –
- **node_to_sparql** (Callable[[*Node*], str]) –
- **returnFormat** (str) –
- **auth** (Optional[Tuple[str, str]]) –

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters
context (Optional[*Graph*]) – a graph instance to query or None

Return type
int

__module__ = 'rdflib.plugins.stores.sparqlstore'

add(*_*, *context=None*, *quoted=False*)

Adds the given statement to a specific context or to the model. The quoted argument is interpreted by formula-aware stores to indicate this statement is quoted/hypothetical It should be an error to not specify a context and have the quoted argument be True. It should also be an error for the quoted argument to be True when the store is not formula-aware.

Parameters

- **_** (Tuple[*Node*, *Node*, *Node*]) –
- **context** (Optional[*Graph*]) –
- **quoted** (bool) –

Return type
None

addN(*quads*)

Adds each item in the list of statements to a specific context. The quoted argument is interpreted by formula-aware stores to indicate this statement is quoted/hypothetical. Note that the default implementation is a redirect to add

Parameters
quads (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
None

add_graph(*graph*)

Add a graph to the store, no effect if the graph already exists. :type graph: *Graph* :param graph: a Graph instance

Return type
None

bind(*prefix*, *namespace*, *override=True*)

Parameters

- **override** (bool) – rebind, even if the given namespace is already bound to another prefix.
- **prefix** (str) –
- **namespace** (*URIRef*) –

Return type
None

commit()

Return type
None

contexts(*triple=None*)

Iterates over results to "SELECT ?NAME { GRAPH ?NAME { ?s ?p ?o } }" or "SELECT ?NAME { GRAPH ?NAME {} }" if triple is None.

Returns instances of this store with the SPARQL wrapper object updated via addNamedGraph(?NAME).

This causes a named-graph-uri key / value pair to be sent over the protocol.

Please note that some SPARQL endpoints are not able to find empty named graphs.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*IdentifiedNode*, None, None]

create(*configuration*)

Parameters
configuration (str) –

Return type
None

destroy(*configuration*)

This destroys the instance of the store identified by the configuration string.

Parameters
configuration (str) –

Return type
None

formula_aware: bool = False

graph_aware: bool = True

namespace(*prefix*)

Parameters
prefix (str) –

Return type
Optional[*URIRef*]

namespaces()

Return type
Iterator[Tuple[str, *URIRef*]]

objects(*subject=None*, *predicate=None*)

A generator of objects with the given subject and predicate

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –

Return type
Generator[*Node*, None, None]

open(*configuration*, *create=False*)

This method is included so that calls to this Store via Graph, e.g. Graph("SPARQLStore"), can set the required parameters

Parameters

- **configuration** (str) –
- **create** (bool) –

Return type
Optional[int]

predicate_objects(*subject=None*)

A generator of (predicate, object) tuples for the given subject

Parameters
subject (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

predicates(*subject=None*, *object=None*)

A generator of predicates with the given subject and object

Parameters

- **subject** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –

Return type
Generator[*Node*, None, None]

prefix(*namespace*)

Parameters
namespace (*URIRef*) –

Return type
Optional[str]

query(*query*, *initNs=None*, *initBindings=None*, *queryGraph=None*, *DEBUG=False*)

If stores provide their own SPARQL implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **query** (Union[*Query*, str]) –
- **initNs** (Optional[Mapping[str, Any]]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –
- **queryGraph** (Optional[str]) –
- **DEBUG** (bool) –

Return type
Result

regex_matching = 0

remove(*_*, *context*)

Remove the set of triples matching the pattern from the store

Parameters

- **_** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type
None

remove_graph(*graph*)

Remove a graph from the store, this should also remove all triples in the graph

Parameters

- **graphid** – a Graph instance
- **graph** (*Graph*) –

Return type
None

rollback()

Return type
None

subject_objects(*predicate=None*)

A generator of (subject, object) tuples for the given predicate

Parameters
predicate (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subject_predicates(*object=None*)

A generator of (subject, predicate) tuples for the given object

Parameters
object (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subjects(*predicate=None*, *object=None*)

A generator of subjects with the given predicate and object

Parameters

- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –

Return type
Generator[*Node*, None, None]

transaction_aware: bool = False

triples(*spo*, *context=None*)

- tuple **(s, o, p)** the triple used as filter for the SPARQL select. (None, None, None) means anything.
- context **context** the graph effectively calling this method.

Returns a tuple of triples executing essentially a SPARQL like SELECT ?subj ?pred ?obj WHERE { ?subj ?pred ?obj }

context may include three parameter to refine the underlying query:

- LIMIT: an integer to limit the number of results
- OFFSET: an integer to enable paging of results
- ORDERBY: an instance of Variable('s'), Variable('o') or Variable('p') or, by default, the first 'None' from the given triple

Warning:

- Using LIMIT or OFFSET automatically include ORDERBY otherwise this is because the results are retrieved in a not deterministic way (depends on the walking path on the graph)
- Using OFFSET without defining LIMIT will discard the first OFFSET - 1 results

```
a_graph.LIMIT = limit
a_graph.OFFSET = offset
triple_generator = a_graph.triples(mytriple):
# do something
```

(continues on next page)

(continued from previous page)

```
# Removes LIMIT and OFFSET if not required for the next triple() calls
del a_graph.LIMIT
del a_graph.OFFSET
```

Parameters

- **spo** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type

Iterator[Tuple[Tuple[*Node*, *Node*, *Node*], None]]

triples_choices(*_*, *context=None*)

A variant of triples that can take a list of terms instead of a single term in any slot. Stores can implement this to optimize the response time from the import default 'fallback' implementation, which will iterate over each term in the list and dispatch to triples.

Parameters

- **_** (Tuple[Union[*Node*, List[*Node*]], Union[*Node*, List[*Node*]], Union[*Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type

Generator[Tuple[Tuple[*Node*, *Node*, *Node*], Iterator[Optional[*Graph*]]], None, None]

update(*query*, *initNs={}*, *initBindings={}*, *queryGraph=None*, *DEBUG=False*)

If stores provide their own (SPARQL) Update implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **query** (Union[*Update*, str]) –
- **initNs** (Dict[str, Any]) –
- **initBindings** (Dict[str, *Identifier*]) –
- **queryGraph** (Optional[*Identifier*]) –
- **DEBUG** (bool) –

Return type

None

class rdflib.plugins.stores.sparqlstore.**SPARQLUpdateStore**(*query_endpoint=None*, *update_endpoint=None*, *sparql11=True*, *context_aware=True*, *postAsEncoded=True*, *autocommit=True*, *dirty_reads=False*, ***kwds*)

Bases: *SPARQLStore*

A store using SPARQL queries for reading and SPARQL Update for changes.

This can be context-aware, if so, any changes will be to the given named graph only.

In favor of the SPARQL 1.1 motivated Dataset, we advise against using this with ConjunctiveGraphs, as it reads and writes from and to the "default graph". Exactly what this means depends on the endpoint and can result in confusion.

For Graph objects, everything works as expected.

See the *SPARQLStore* base class for more information.

Parameters

- **query_endpoint** (Optional[str]) –
- **update_endpoint** (Optional[str]) –
- **sparql11** (bool) –
- **context_aware** (bool) –
- **postAsEncoded** (bool) –
- **autocommit** (bool) –
- **dirty_reads** (bool) –

```
BLOCK_END = '}'

BLOCK_FINDING_PATTERN = re.compile('(?P<block_start>{)|(?P<block_end>})|(?
P<block_content>((\'([^\'\\\\]|\\\\.)*\')|("([^"\\\\]|\\\\.)*")|(\'\'\'((\'|\'\')?
([^\'\\\\]|\\\\.))*\'\'\')|("""(("|"")?([^"\\\\]|\\\\.))*"""))|(<([^<>"{}|^`\\])

BLOCK_START = '{'

BlockContent = '((\'([^\'\\\\]|\\\\.)*\')|("([^"\\\\]|\\\\.)*")|(\'\'\'((\'|\'\')?
([^\'\\\\]|\\\\.))*\'\'\')|("""(("|"")?([^"\\\\]|\\\\.))*"""))|(<([^<>"{}|^`\\]\\\\
[\\x00-\\x20])*>)|(#[^\\x0D\\x0A]*([\\x0D\\x0A]|\\Z))|(\\\\.)'

BlockFinding = '(?P<block_start>{)|(?P<block_end>})|(?P<block_content>((\'([^\'\\\\
]|\\\\.)*\')|("([^"\\\\]|\\\\.)*")|(\'\'\'((\'|\'\')?([^\'\\\\]|\\\\.))*\'\'\'
')|("""(("|"")?([^"\\\\]|\\\\.))*"""))|(<([^<>"{}|^`\\]\\\\[\\x00-\\x20])*>)|(#[^\\
x0D\\x0A]*([\\x0D\\x0A]|\\Z))|(\\\\.))'

COMMENT = '#[^\\x0D\\x0A]*([\\x0D\\x0A]|\\Z)'

ESCAPED = '\\\\.'

IRIREF = '<([^<>"{}|^`\\]\\\\[\\x00-\\x20])*>'

STRING_LITERAL1 = "'([^'\\\\]|\\\\.)*'"

STRING_LITERAL2 = '"([^"\\\\]|\\\\.)*"'

STRING_LITERAL_LONG1 = "'''(('|'')?([^'\\\\]|\\\\.))*'''"

STRING_LITERAL_LONG2 = '"""(("|"")?([^"\\\\]|\\\\.))*"""'

String = '(\'([^\'\\\\]|\\\\.)*\')|("([^"\\\\]|\\\\.)*")|(\'\'\'((\'|\'\')?([^\'\\\\
]|\\\\.))*\'\'\')|("""(("|"")?([^"\\\\]|\\\\.))*""")'
```

__init__(*query_endpoint=None*, *update_endpoint=None*, *sparql11=True*, *context_aware=True*, *postAsEncoded=True*, *autocommit=True*, *dirty_reads=False*, ***kwds*)

:param autocommit if set, the store will commit after every writing operations. If False, we only make queries on the server once commit is called.

:param dirty_reads if set, we do not commit before reading. So you cannot read what you wrote before manually calling commit.

Parameters

- **query_endpoint** (Optional[str]) –
- **update_endpoint** (Optional[str]) –
- **sparql11** (bool) –
- **context_aware** (bool) –
- **postAsEncoded** (bool) –
- **autocommit** (bool) –
- **dirty_reads** (bool) –

__len__(**args*, ***kwargs*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters

- **context** – a graph instance to query or None
- **args** (Any) –
- **kwargs** (Any) –

Return type
int

__module__ = 'rdflib.plugins.stores.sparqlstore'

add(*spo*, *context=None*, *quoted=False*)

Add a triple to the store of triples.

Parameters

- **spo** (Tuple[*Node*, *Node*, *Node*]) –
- **context** (Optional[*Graph*]) –
- **quoted** (bool) –

Return type
None

addN(*quads*)

Add a list of quads to the store.

Parameters
quads (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
None

add_graph(*graph*)

Add a graph to the store, no effect if the graph already exists. :type graph: *Graph* :param graph: a Graph instance

Return type
None

commit()

add(), addN(), and remove() are transactional to reduce overhead of many small edits. Read and update() calls will automatically commit any outstanding edits. This should behave as expected most of the time, except that alternating writes and reads can degenerate to the original call-per-triple situation that originally existed.

Return type
None

contexts(**args*, ***kwargs*)

Iterates over results to "SELECT ?NAME { GRAPH ?NAME { ?s ?p ?o } }" or "SELECT ?NAME { GRAPH ?NAME {} }" if triple is None.

Returns instances of this store with the SPARQL wrapper object updated via addNamedGraph(?NAME).

This causes a named-graph-uri key / value pair to be sent over the protocol.

Please note that some SPARQL endpoints are not able to find empty named graphs.

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type
Generator[*IdentifiedNode*, None, None]

nsBindings: Dict[str, Any]

objects(*subject=None*, *predicate=None*)

A generator of objects with the given subject and predicate

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –

Return type
Generator[*Node*, None, None]

open(*configuration*, *create=False*)

sets the endpoint URLs for this SPARQLStore

Parameters

- **configuration** (Union[str, Tuple[str, str]]) – either a tuple of (query_endpoint, update_endpoint), or a string with the endpoint which is configured as query and update endpoint
- **create** (bool) – if True an exception is thrown.

Return type
None

predicate_objects(*subject=None*)

A generator of (predicate, object) tuples for the given subject

Parameters
subject (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

predicates(*subject=None*, *object=None*)

A generator of predicates with the given subject and object

Parameters

- **subject** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –

Return type
Generator[*Node*, None, None]

query(**args*, ***kwargs*)

If stores provide their own SPARQL implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type
Result

remove(*spo*, *context*)

Remove a triple from the store

Parameters

- **spo** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type
None

remove_graph(*graph*)

Remove a graph from the store, this should also remove all triples in the graph

Parameters

- **graphid** – a Graph instance
- **graph** (*Graph*) –

Return type
None

rollback()

Return type
None

setTimeout(*timeout*)

Return type
None

subject_objects(*predicate=None*)

A generator of (subject, object) tuples for the given predicate

Parameters
predicate (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subject_predicates(*object=None*)

A generator of (subject, predicate) tuples for the given object

Parameters
object (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subjects(*predicate=None*, *object=None*)

A generator of subjects with the given predicate and object

Parameters

- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –

Return type
Generator[*Node*, None, None]

triples(**args*, ***kwargs*)

- tuple **(s, o, p)** the triple used as filter for the SPARQL select. (None, None, None) means anything.
- context **context** the graph effectively calling this method.

Returns a tuple of triples executing essentially a SPARQL like SELECT ?subj ?pred ?obj WHERE { ?subj ?pred ?obj }

context may include three parameter to refine the underlying query:

- LIMIT: an integer to limit the number of results
- OFFSET: an integer to enable paging of results
- ORDERBY: an instance of Variable('s'), Variable('o') or Variable('p') or, by default, the first 'None' from the given triple

Warning:

- Using LIMIT or OFFSET automatically include ORDERBY otherwise this is because the results are retrieved in a not deterministic way (depends on the walking path on the graph)

- Using OFFSET without defining LIMIT will discard the first OFFSET - 1 results

```
a_graph.LIMIT = limit
a_graph.OFFSET = offset
triple_generator = a_graph.triples(mytriple):
# do something
# Removes LIMIT and OFFSET if not required for the next triple() calls
del a_graph.LIMIT
del a_graph.OFFSET
```

Parameters

- **args** (Any) –
- **kwargs** (Any) –

Return type

Iterator[Tuple[Tuple[*Node*, *Node*, *Node*], None]]

update(*query*, *initNs={}*, *initBindings={}*, *queryGraph=None*, *DEBUG=False*)

Perform a SPARQL Update Query against the endpoint, INSERT, LOAD, DELETE etc. Setting initNs adds PREFIX declarations to the beginning of the update. Setting initBindings adds inline VALUEs to the beginning of every WHERE clause. By the SPARQL grammar, all operations that support variables (namely INSERT and DELETE) require a WHERE clause. Important: initBindings fails if the update contains the substring 'WHERE {' which does not denote a WHERE clause, e.g. if it is part of a literal.

Context-aware query rewriting

- **When:** If context-awareness is enabled and the graph is not the default graph of the store.
- **Why:** To ensure consistency with the *Memory* store. The graph must accept "local" SPARQL requests (requests with no GRAPH keyword) as if it was the default graph.
- **What is done:** These "local" queries are rewritten by this store. The content of each block of a SPARQL Update operation is wrapped in a GRAPH block except if the block is empty. This basically causes INSERT, INSERT DATA, DELETE, DELETE DATA and WHERE to operate only on the context.
- **Example:** `"INSERT DATA { <urn:michel> <urn:likes> <urn:pizza> }"` is converted into `"INSERT DATA { GRAPH <urn:graph> { <urn:michel> <urn:likes> <urn:pizza> } }"`.
- **Warning:** Queries are presumed to be "local" but this assumption is **not checked**. For instance, if the query already contains GRAPH blocks, the latter will be wrapped in new GRAPH blocks.
- **Warning:** A simplified grammar is used that should tolerate extensions of the SPARQL grammar. Still, the process may fail in uncommon situations and produce invalid output.

Parameters

- **query** (Union[*Update*, str]) –
- **initNs** (Dict[str, Any]) –
- **initBindings** (Dict[str, *Identifier*]) –
- **queryGraph** (Optional[str]) –

- **DEBUG** (bool) –

```
where_pattern = re.compile('(?P<where>WHERE\\s*\\{)', re.IGNORECASE)
```

Module contents

This package contains modules for additional RDFLib stores

Module contents

Default plugins for rdflib.

This is a namespace package and contains the default plugins for rdflib.

rdflib.tools package

Submodules

rdflib.tools.chunk_serializer module

This file provides a single function *serialize_in_chunks()* which can serialize a Graph into a number of NT files with a maximum number of triples or maximum file size.

There is an option to preserve any prefixes declared for the original graph in the first file, which will be a Turtle file.

rdflib.tools.chunk_serializer.**serialize_in_chunks**(*g*, *max_triples=10000*, *max_file_size_kb=None*, *file_name_stem='chunk'*, *output_dir=None*, *write_prefixes=False*)

Serializes a given Graph into a series of n-triples with a given length.

Parameters

- **g** (*Graph*) – The graph to serialize.
- **max_file_size_kb** (Optional[int]) – Maximum size per NT file in kB (1,000 bytes) Equivalent to ~6,000 triples, depending on Literal sizes.
- **max_triples** (int) – Maximum size per NT file in triples Equivalent to lines in file.

  If both this parameter and max_file_size_kb are set, max_file_size_kb will be used.
- **file_name_stem** (str) – Prefix of each file name. e.g. "chunk" = chunk_000001.nt, chunk_000002.nt…
- **output_dir** (Optional[Path]) – The directory you want the files to be written to.
- **write_prefixes** (bool) – The first file created is a Turtle file containing original graph prefixes.

See `../test/test_tools/test_chunk_serializer.py` for examples of this in use.

Return type

None

rdflib.tools.csv2rdf module

A commandline tool for semi-automatically converting CSV to RDF.

See also https://github.com/RDFLib/pyTARQL in the RDFlib family of tools

try: `csv2rdf --help`

class rdflib.tools.csv2rdf.**CSV2RDF**

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.tools.csv2rdf', '__init__': <function CSV2RDF.__init__>, 'triple': <function CSV2RDF.triple>, 'convert': <function CSV2RDF.convert>, '__dict__': <attribute '__dict__' of 'CSV2RDF' objects>, '__weakref__': <attribute '__weakref__' of 'CSV2RDF' objects>, '__doc__': None, '__annotations__': {}})

__init__()

__module__ = 'rdflib.tools.csv2rdf'

__weakref__

list of weak references to the object (if defined)

convert(*csvreader*)

triple(*s*, *p*, *o*)

rdflib.tools.defined_namespace_creator module

rdflib.tools.defined_namespace_creator.**get_target_namespace_elements**(*g*, *target_namespace*)

Parameters

- **g** (*Graph*) –
- **target_namespace** (str) –

Return type

Tuple[List[Tuple[str, str]], List[str]]

rdflib.tools.defined_namespace_creator.**make_dn_file**(*output_file_name*, *target_namespace*, *elements_strs*, *object_id*, *fail*)

Parameters

- **output_file_name** (Path) –
- **target_namespace** (str) –
- **elements_strs** (Iterable[str]) –
- **object_id** (str) –
- **fail** (bool) –

Return type

None

rdflib.tools.defined_namespace_creator.**validate_namespace**(*namespace*)

Parameters
namespace (str) –

Return type
None

rdflib.tools.defined_namespace_creator.**validate_object_id**(*object_id*)

Parameters
object_id (str) –

Return type
None

rdflib.tools.graphisomorphism module

A commandline tool for testing if RDF graphs are isomorpic, i.e. equal if BNode labels are ignored.

class rdflib.tools.graphisomorphism.**IsomorphicTestableGraph**(***kargs*)

Bases: *Graph*

Ported from: http://www.w3.org/2001/sw/DataAccess/proto-tests/tools/rdfdiff.py (Sean B Palmer's RDF Graph Isomorphism Tester)

__eq__(*G*)

Graph isomorphism testing.

__hash__ = None

__init__(***kargs*)

__module__ = 'rdflib.tools.graphisomorphism'

__ne__(*G*)

Negative graph isomorphism testing.

hashtriples()

internal_hash()

This is defined instead of __hash__ to avoid a circular recursion scenario with the Memory store for rdflib which requires a hash lookup in order to return a generator of triples

vhash(*term*, *done=False*)

vhashtriple(*triple*, *term*, *done*)

vhashtriples(*term*, *done*)

rdflib.tools.graphisomorphism.**main**()

rdflib.tools.rdf2dot module

A commandline tool for drawing RDF graphs in Graphviz DOT format

You can draw the graph of an RDF file directly:

rdflib.tools.rdf2dot.**main**()

rdflib.tools.rdf2dot.**rdf2dot**(*g*, *stream*, *opts={}*)

Convert the RDF graph to DOT writes the dot output to the stream

rdflib.tools.rdfpipe module

A commandline tool for parsing RDF in different formats and serializing the resulting graph to a chosen format.

rdflib.tools.rdfpipe.**main**()

rdflib.tools.rdfpipe.**make_option_parser**()

rdflib.tools.rdfpipe.**parse_and_serialize**(*input_files*, *input_format*, *guess*, *outfile*, *output_format*, *ns_bindings*, *store_conn=''*, *store_type=None*)

rdflib.tools.rdfs2dot module

A commandline tool for drawing RDFS Class diagrams in Graphviz DOT format

You can draw the graph of an RDFS file directly:

rdflib.tools.rdfs2dot.**main**()

rdflib.tools.rdfs2dot.**rdfs2dot**(*g*, *stream*, *opts={}*)

Convert the RDFS schema in a graph writes the dot output to the stream

Module contents

Various commandline tools for working with RDFLib

Submodules

rdflib.collection module

class rdflib.collection.**Collection**(*graph*, *uri*, *seq=[]*)

Bases: `object`

See "Emulating container types": https://docs.python.org/reference/datamodel.html#emulating-container-types

```
>>> from rdflib.term import Literal
>>> from rdflib.graph import Graph
>>> from pprint import pprint
>>> listname = BNode()
>>> g = Graph('Memory')
>>> listItem1 = BNode()
```

(continues on next page)

(continued from previous page)

```
>>> listItem2 = BNode()
>>> g.add((listname, RDF.first, Literal(1)))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listname, RDF.rest, listItem1))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem1, RDF.first, Literal(2)))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem1, RDF.rest, listItem2))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem2, RDF.rest, RDF.nil))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem2, RDF.first, Literal(3)))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> c = Collection(g,listname)
>>> pprint([term.n3() for term in c])
[u'"1"^^<http://www.w3.org/2001/XMLSchema#integer>',
 u'"2"^^<http://www.w3.org/2001/XMLSchema#integer>',
 u'"3"^^<http://www.w3.org/2001/XMLSchema#integer>']
```

```
>>> Literal(1) in c
True
>>> len(c)
3
>>> c._get_container(1) == listItem1
True
>>> c.index(Literal(2)) == 1
True
```

Parameters

- **graph** (*Graph*) –
- **uri** (*Node*) –
- **seq** (List[*Node*]) –

__delitem__(*key*)

```
>>> from rdflib.namespace import RDF, RDFS
>>> from rdflib import Graph
>>> from pprint import pformat
>>> g = Graph()
>>> a = BNode('foo')
>>> b = BNode('bar')
>>> c = BNode('baz')
>>> g.add((a, RDF.first, RDF.type))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((a, RDF.rest, b))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((b, RDF.first, RDFS.label))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((b, RDF.rest, c))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
```

(continues on next page)

(continued from previous page)

```
>>> g.add((c, RDF.first, RDFS.comment))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((c, RDF.rest, RDF.nil))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> len(g)
6
>>> def listAncestry(node, graph):
...   for i in graph.subjects(RDF.rest, node):
...     yield i
>>> [str(node.n3())
...   for node in g.transitiveClosure(listAncestry, RDF.nil)]
['_:baz', '_:bar', '_:foo']
>>> lst = Collection(g, a)
>>> len(lst)
3
>>> b == lst._get_container(1)
True
>>> c == lst._get_container(2)
True
>>> del lst[1]
>>> len(lst)
2
>>> len(g)
4
```

Parameters
key (int) –

Return type
None

__dict__ = mappingproxy({'__module__': 'rdflib.collection', '__doc__': '\n See "Emulating container types":\n https://docs.python.org/reference/datamodel.html#emulating-container-types\n\n >>> from rdflib.term import Literal\n >>> from rdflib.graph import Graph\n >>> from pprint import pprint\n >>> listname = BNode()\n >>> g = Graph(\'Memory\')\n >>> listItem1 = BNode()\n >>> listItem2 = BNode()\n >>> g.add((listname, RDF.first, Literal(1))) # doctest: +ELLIPSIS\n <Graph identifier=... (<class \'rdflib.graph.Graph\'>)>\n >>> g.add((listname, RDF.rest, listItem1)) # doctest: +ELLIPSIS\n <Graph identifier=... (<class \'rdflib.graph.Graph\'>)>\n >>> g.add((listItem1, RDF.first, Literal(2))) # doctest: +ELLIPSIS\n <Graph identifier=... (<class \'rdflib.graph.Graph\'>)>\n >>> g.add((listItem1, RDF.rest, listItem2)) # doctest: +ELLIPSIS\n <Graph identifier=... (<class \'rdflib.graph.Graph\'>)>\n >>> g.add((listItem2, RDF.rest, RDF.nil)) # doctest: +ELLIPSIS\n <Graph identifier=... (<class \'rdflib.graph.Graph\'>)>\n >>> g.add((listItem2, RDF.first, Literal(3))) # doctest: +ELLIPSIS\n <Graph identifier=... (<class \'rdflib.graph.Graph\'>)>\n >>> c = Collection(g,listname)\n >>> pprint([term.n3() for term in c])\n [u\'"1"^^<http://www.w3.org/2001/XMLSchema#integer>\',\n u\'"2"^^<http://www.w3.org/2001/XMLSchema#integer>\',\n u\'"3"^^<http://www.w3.org/2001/XMLSchema#integer>\']\n\n >>> Literal(1) in c\n True\n >>> len(c)\n 3\n >>> c._get_container(1) == listItem1\n True\n >>> c.index(Literal(2)) == 1\n True\n ', '__init__': <function Collection.__init__>, 'n3': <function Collection.n3>, '_get_container': <function Collection._get_container>, '__len__': <function Collection.__len__>, 'index': <function Collection.index>, '__getitem__': <function Collection.__getitem__>, '__setitem__': <function Collection.__setitem__>, '__delitem__': <function Collection.__delitem__>, '__iter__': <function Collection.__iter__>, '_end': <function Collection._end>, 'append': <function Collection.append>, '__iadd__': <function Collection.__iadd__>, 'clear': <function Collection.clear>, '__dict__': <attribute '__dict__' of 'Collection' objects>, '__weakref__': <attribute '__weakref__' of 'Collection' objects>, '__annotations__': {}})

__getitem__(*key*)

TODO

Parameters
key (int) –

Return type
Node

__iadd__(*other*)

Parameters
other (Iterable[*Node*]) –

__init__(*graph*, *uri*, *seq=[]*)

Parameters

- **graph** (*Graph*) –
- **uri** (*Node*) –
- **seq** (List[*Node*]) –

__iter__()

Iterator over items in Collections

Return type
Iterator[*Node*]

__len__()
length of items in collection.

Return type
int

__module__ = 'rdflib.collection'

__setitem__(*key*, *value*)
TODO

Parameters
- **key** (int) –
- **value** (*Node*) –

Return type
None

__weakref__
list of weak references to the object (if defined)

append(*item*)

```
>>> from rdflib.term import Literal
>>> from rdflib.graph import Graph
>>> listname = BNode()
>>> g = Graph()
>>> c = Collection(g,listname,[Literal(1),Literal(2)])
>>> links = [
...     list(g.subjects(object=i, predicate=RDF.first))[0] for i in c]
>>> len([i for i in links if (i, RDF.rest, RDF.nil) in g])
1
```

Parameters
item (*Node*) –

Return type
Collection

clear()

index(*item*)
Returns the 0-based numerical index of the item in the list

Parameters
item (*Node*) –

Return type
int

n3()

```
>>> from rdflib.term import Literal
>>> from rdflib.graph import Graph
>>> listname = BNode()
>>> g = Graph('Memory')
>>> listItem1 = BNode()
>>> listItem2 = BNode()
>>> g.add((listname, RDF.first, Literal(1)))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listname, RDF.rest, listItem1))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem1, RDF.first, Literal(2)))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem1, RDF.rest, listItem2))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem2, RDF.rest, RDF.nil))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((listItem2, RDF.first, Literal(3)))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> c = Collection(g, listname)
>>> print(c.n3())
( "1"^^<http://www.w3.org/2001/XMLSchema#integer>
  "2"^^<http://www.w3.org/2001/XMLSchema#integer>
  "3"^^<http://www.w3.org/2001/XMLSchema#integer> )
```

Return type

str

rdflib.compare module

A collection of utilities for canonicalizing and inspecting graphs.

Among other things, they solve of the problem of deterministic bnode comparisons.

Warning: the time to canonicalize bnodes may increase exponentially on degenerate larger graphs. Use with care!

Example of comparing two graphs:

```
>>> g1 = Graph().parse(format='n3', data='''
...     @prefix : <http://example.org/ns#> .
...     <http://example.org> :rel
...         <http://example.org/same>,
...         [ :label "Same" ],
...         <http://example.org/a>,
...         [ :label "A" ] .
... ''')
>>> g2 = Graph().parse(format='n3', data='''
...     @prefix : <http://example.org/ns#> .
...     <http://example.org> :rel
...         <http://example.org/same>,
...         [ :label "Same" ],
...         <http://example.org/b>,
...         [ :label "B" ] .
... ''')
```

(continues on next page)

(continued from previous page)

```
>>>
>>> iso1 = to_isomorphic(g1)
>>> iso2 = to_isomorphic(g2)
```

These are not isomorphic:

```
>>> iso1 == iso2
False
```

Diff the two graphs:

```
>>> in_both, in_first, in_second = graph_diff(iso1, iso2)
```

Present in both:

```
>>> def dump_nt_sorted(g):
...     for l in sorted(g.serialize(format='nt').splitlines()):
...         if l: print(l.decode('ascii'))

>>> dump_nt_sorted(in_both)
<http://example.org>
    <http://example.org/ns#rel> <http://example.org/same> .
<http://example.org>
    <http://example.org/ns#rel> _:cbcaabaaba17fecbc304a64f8edee4335e .
_:cbcaabaaba17fecbc304a64f8edee4335e
    <http://example.org/ns#label> "Same" .
```

Only in first:

```
>>> dump_nt_sorted(in_first)
<http://example.org>
    <http://example.org/ns#rel> <http://example.org/a> .
<http://example.org>
    <http://example.org/ns#rel> _:cb124e4c6da0579f810c0ffe4eff485bd9 .
_:cb124e4c6da0579f810c0ffe4eff485bd9
    <http://example.org/ns#label> "A" .
```

Only in second:

```
>>> dump_nt_sorted(in_second)
<http://example.org>
    <http://example.org/ns#rel> <http://example.org/b> .
<http://example.org>
    <http://example.org/ns#rel> _:cb558f30e21ddfc05ca53108348338ade8 .
_:cb558f30e21ddfc05ca53108348338ade8
    <http://example.org/ns#label> "B" .
```

class rdflib.compare.**IsomorphicGraph**(***kwargs*)

Bases: *ConjunctiveGraph*

An implementation of the RGDA1 graph digest algorithm.

An implementation of RGDA1 (publication below), a combination of Sayers & Karp's graph digest algorithm using sum and SHA-256 <http://www.hpl.hp.com/techreports/2003/HPL-2003-235R1.pdf> and traces <http://pallini.di.uniroma1.it>, an average case polynomial time algorithm for graph canonicalization.

McCusker, J. P. (2015). WebSig: A Digital Signature Framework for the Web. Rensselaer Polytechnic Institute, Troy, NY. http://gradworks.umi.com/3727015.pdf

__eq__(*other*)

Graph isomorphism testing.

__hash__()

Return hash(self).

__init__(***kwargs*)

__module__ = 'rdflib.compare'

__ne__(*other*)

Negative graph isomorphism testing.

graph_digest(*stats=None*)

Synonym for IsomorphicGraph.internal_hash.

internal_hash(*stats=None*)

This is defined instead of __hash__ to avoid a circular recursion scenario with the Memory store for rdflib which requires a hash lookup in order to return a generator of triples.

rdflib.compare.**graph_diff**(*g1*, *g2*)

Returns three sets of triples: "in both", "in first" and "in second".

Parameters

- **g1** (*Graph*) –
- **g2** (*Graph*) –

Return type

Tuple[*Graph*, *Graph*, *Graph*]

rdflib.compare.**isomorphic**(*graph1*, *graph2*)

Compare graph for equality.

Uses an algorithm to compute unique hashes which takes bnodes into account.

Examples:

```
>>> g1 = Graph().parse(format='n3', data='''
...     @prefix : <http://example.org/ns#> .
...     <http://example.org> :rel <http://example.org/a> .
...     <http://example.org> :rel <http://example.org/b> .
...     <http://example.org> :rel [ :label "A bnode." ] .
... ''')
>>> g2 = Graph().parse(format='n3', data='''
...     @prefix ns: <http://example.org/ns#> .
...     <http://example.org> ns:rel [ ns:label "A bnode." ] .
...     <http://example.org> ns:rel <http://example.org/b>,
...             <http://example.org/a> .
... ''')
>>> isomorphic(g1, g2)
True

>>> g3 = Graph().parse(format='n3', data='''
...     @prefix : <http://example.org/ns#> .
```

(continues on next page)

(continued from previous page)

```
...     <http://example.org> :rel <http://example.org/a> .
...     <http://example.org> :rel <http://example.org/b> .
...     <http://example.org> :rel <http://example.org/c> .
... ''')
>>> isomorphic(g1, g3)
False
```

Parameters

- **graph1** (*Graph*) –
- **graph2** (*Graph*) –

Return type
bool

rdflib.compare.**similar**(*g1*, *g2*)

Checks if the two graphs are "similar".

Checks if the two graphs are "similar", by comparing sorted triples where all bnodes have been replaced by a singular mock bnode (the `_MOCK_BNODE`).

This is a much cheaper, but less reliable, alternative to the comparison algorithm in `isomorphic`.

Parameters

- **g1** (*Graph*) –
- **g2** (*Graph*) –

rdflib.compare.**to_canonical_graph**(*g1*, *stats=None*)

Creates a canonical, read-only graph.

Creates a canonical, read-only graph where all bnode id:s are based on deterministical SHA-256 checksums, correlated with the graph contents.

Parameters

- **g1** (*Graph*) –
- **stats** (Optional[Dict[str, Union[int, str]]]) –

Return type
ReadOnlyGraphAggregate

rdflib.compare.**to_isomorphic**(*graph*)

Parameters
graph (*Graph*) –

Return type
IsomorphicGraph

rdflib.compat module

Utility functions and objects to ease Python 2/3 compatibility, and different versions of support libraries.

rdflib.compat.**ascii**(*stream*)

rdflib.compat.**bopen**(**args*, ***kwargs*)

rdflib.compat.**cast_bytes**(*s*, *enc='utf-8'*)

rdflib.compat.**decodeStringEscape**(*s*)

rdflib.compat.**decodeUnicodeEscape**(*escaped*)

Parameters
escaped (str) –

Return type
str

rdflib.compat.**sign**(*n*)

rdflib.container module

class rdflib.container.**Alt**(*graph*, *uri*, *seq=[]*)

Bases: *Container*

__init__(*graph*, *uri*, *seq=[]*)

Creates a Container

Parameters

- **graph** – a Graph instance
- **uri** – URI or Blank Node of the Container
- **seq** – the elements of the Container
- **rtype** – the type of Container, one of “Bag”, “Seq” or “Alt”

__module__ = 'rdflib.container'

anyone()

class rdflib.container.**Bag**(*graph*, *uri*, *seq=[]*)

Bases: *Container*

Unordered container (no preference order of elements)

__init__(*graph*, *uri*, *seq=[]*)

Creates a Container

Parameters

- **graph** – a Graph instance
- **uri** – URI or Blank Node of the Container
- **seq** – the elements of the Container
- **rtype** – the type of Container, one of “Bag”, “Seq” or “Alt”

__module__ = 'rdflib.container'

class rdflib.container.**Container**(*graph*, *uri*, *seq=[]*, *rtype='Bag'*)

Bases: object

A class for constructing RDF containers, as per https://www.w3.org/TR/rdf11-mt/#rdf-containers

Basic usage, creating a Bag and adding to it:

```
>>> from rdflib import Graph, BNode, Literal, Bag
>>> g = Graph()
>>> b = Bag(g, BNode(), [Literal("One"), Literal("Two"), Literal("Three")])
>>> print(g.serialize(format="turtle"))
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[] a rdf:Bag ;
    rdf:_1 "One" ;
    rdf:_2 "Two" ;
    rdf:_3 "Three" .



>>> # print out an item using an index reference
>>> print(b[2])
Two

>>> # add a new item
>>> b.append(Literal("Hello"))
<rdflib.container.Bag object at ...>
>>> print(g.serialize(format="turtle"))
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[] a rdf:Bag ;
    rdf:_1 "One" ;
    rdf:_2 "Two" ;
    rdf:_3 "Three" ;
    rdf:_4 "Hello" .
```

__delitem__(*key*)

Removing the item with index key or predicate rdf:_key

__dict__ = mappingproxy({'__module__': 'rdflib.container', '__doc__': 'A class for constructing RDF containers, as per https://www.w3.org/TR/rdf11-mt/#rdf-containers\n\n Basic usage, creating a ``Bag`` and adding to it::\n\n >>> from rdflib import Graph, BNode, Literal, Bag\n >>> g = Graph()\n >>> b = Bag(g, BNode(), [Literal("One"), Literal("Two"), Literal("Three")])\n >>> print(g.serialize(format="turtle"))\n @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .\n <BLANKLINE>\n [] a rdf:Bag ;\n rdf:_1 "One" ;\n rdf:_2 "Two" ;\n rdf:_3 "Three" .\n <BLANKLINE>\n <BLANKLINE>\n\n >>> # print out an item using an index reference\n >>> print(b[2])\n Two\n\n >>> # add a new item\n >>> b.append(Literal("Hello")) # doctest: +ELLIPSIS\n <rdflib.container.Bag object at ...>\n >>> print(g.serialize(format="turtle"))\n @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .\n <BLANKLINE>\n [] a rdf:Bag ;\n rdf:_1 "One" ;\n rdf:_2 "Two" ;\n rdf:_3 "Three" ;\n rdf:_4 "Hello" .\n <BLANKLINE>\n <BLANKLINE>\n\n ', '__init__': <function Container.__init__>, 'n3': <function Container.n3>, '_get_container': <function Container._get_container>, '__len__': <function Container.__len__>, 'type_of_conatiner': <function Container.type_of_conatiner>, 'index': <function Container.index>, '__getitem__': <function Container.__getitem__>, '__setitem__': <function Container.__setitem__>, '__delitem__': <function Container.__delitem__>, 'items': <function Container.items>, 'end': <function Container.end>, 'append': <function Container.append>, 'append_multiple': <function Container.append_multiple>, 'clear': <function Container.clear>, '__dict__': <attribute '__dict__' of 'Container' objects>, '__weakref__': <attribute '__weakref__' of 'Container' objects>, '__annotations__': {}})

__getitem__(*key*)

Returns item of the container at index key

__init__(*graph*, *uri*, *seq=[]*, *rtype='Bag'*)

Creates a Container

Parameters

- **graph** – a Graph instance
- **uri** – URI or Blank Node of the Container
- **seq** – the elements of the Container
- **rtype** – the type of Container, one of "Bag", "Seq" or "Alt"

__len__()

Number of items in container

__module__ = 'rdflib.container'

__setitem__(*key*, *value*)

Sets the item at index key or predicate rdf:_key of the container to value

__weakref__

list of weak references to the object (if defined)

append(*item*)

Adding item to the end of the container

append_multiple(*other*)

Adding multiple elements to the container to the end which are in python list other

clear()

Removing all elements from the container

end()

index(*item*)

Returns the 1-based numerical index of the item in the container

items()

Returns a list of all items in the container

n3()

type_of_conatiner()

exception rdflib.container.**NoElementException**(*message='rdf:Alt Container is empty'*)

Bases: Exception

__init__(*message='rdf:Alt Container is empty'*)

__module__ = 'rdflib.container'

__str__()

Return str(self).

__weakref__

list of weak references to the object (if defined)

class rdflib.container.**Seq**(*graph*, *uri*, *seq=[]*)

Bases: *Container*

__init__(*graph*, *uri*, *seq=[]*)

Creates a Container

Parameters

- **graph** – a Graph instance
- **uri** – URI or Blank Node of the Container
- **seq** – the elements of the Container
- **rtype** – the type of Container, one of “Bag”, “Seq” or “Alt”

__module__ = 'rdflib.container'

add_at_position(*pos*, *item*)

rdflib.events module

Dirt Simple Events

A Dispatcher (or a subclass of Dispatcher) stores event handlers that are ‘fired’ simple event objects when interesting things happen.

Create a dispatcher:

```
>>> d = Dispatcher()
```

Now create a handler for the event and subscribe it to the dispatcher to handle Event events. A handler is a simple function or method that accepts the event as an argument:

```
>>> def handler1(event): print(repr(event))
>>> d.subscribe(Event, handler1)
<rdflib.events.Dispatcher object at ...>
```

Now dispatch a new event into the dispatcher, and see handler1 get fired:

```
>>> d.dispatch(Event(foo='bar', data='yours', used_by='the event handlers'))
<rdflib.events.Event ['data', 'foo', 'used_by']>
```

class rdflib.events.**Dispatcher**

Bases: object

An object that can dispatch events to a privately managed group of subscribers.

__dict__ = mappingproxy({'__module__': 'rdflib.events', '__doc__': '\n An object that can dispatch events to a privately managed group of\n subscribers.\n ', '_dispatch_map': None, 'set_map': <function Dispatcher.set_map>, 'get_map': <function Dispatcher.get_map>, 'subscribe': <function Dispatcher.subscribe>, 'dispatch': <function Dispatcher.dispatch>, '__dict__': <attribute '__dict__' of 'Dispatcher' objects>, '__weakref__': <attribute '__weakref__' of 'Dispatcher' objects>, '__annotations__': {}})

__module__ = 'rdflib.events'

__weakref__

list of weak references to the object (if defined)

dispatch(*event*)

Dispatch the given event to the subscribed handlers for the event's type

get_map()

set_map(*amap*)

subscribe(*event_type*, *handler*)

Subscribe the given handler to an event_type. Handlers are called in the order they are subscribed.

class rdflib.events.**Event**(***kw*)

Bases: object

An event is a container for attributes. The source of an event creates this object, or a subclass, gives it any kind of data that the events handlers need to handle the event, and then calls notify(event).

The target of an event registers a function to handle the event it is interested with subscribe(). When a sources calls notify(event), each subscriber to that event will be called in no particular order.

__dict__ = mappingproxy({'__module__': 'rdflib.events', '__doc__': '\n An event is a container for attributes. The source of an event\n creates this object, or a subclass, gives it any kind of data that\n the events handlers need to handle the event, and then calls\n notify(event).\n\n The target of an event registers a function to handle the event it\n is interested with subscribe(). When a sources calls\n notify(event), each subscriber to that event will be called in no\n particular order.\n ', '__init__': <function Event.__init__>, '__repr__': <function Event.__repr__>, '__dict__': <attribute '__dict__' of 'Event' objects>, '__weakref__': <attribute '__weakref__' of 'Event' objects>, '__annotations__': {}})

__init__(**kw)

__module__ = 'rdflib.events'

__repr__()

Return repr(self).

__weakref__

list of weak references to the object (if defined)

rdflib.exceptions module

TODO:

exception rdflib.exceptions.Error(msg=None)

Bases: Exception

Base class for rdflib exceptions.

Parameters
msg (Optional[str]) –

__init__(msg=None)

Parameters
msg (Optional[str]) –

__module__ = 'rdflib.exceptions'

__weakref__

list of weak references to the object (if defined)

exception rdflib.exceptions.ParserError(msg)

Bases: *Error*

RDF Parser error.

Parameters
msg (str) –

__init__(msg)

Parameters
msg (str) –

__module__ = 'rdflib.exceptions'

__str__()

Return str(self).

Return type
str

exception rdflib.exceptions.UniquenessError(values)

Bases: *Error*

A uniqueness assumption was made in the context, and that is not true

Parameters
values (Any) –

__init__(*values*)

Parameters
values (Any) –

__module__ = 'rdflib.exceptions'

rdflib.graph module

RDFLib defines the following kinds of Graphs:

- *Graph*
- *QuotedGraph*
- *ConjunctiveGraph*
- *Dataset*

Graph

An RDF graph is a set of RDF triples. Graphs support the python `in` operator, as well as iteration and some operations like union, difference and intersection.

see *Graph*

Conjunctive Graph

A Conjunctive Graph is the most relevant collection of graphs that are considered to be the boundary for closed world assumptions. This boundary is equivalent to that of the store instance (which is itself uniquely identified and distinct from other instances of *Store* that signify other Conjunctive Graphs). It is equivalent to all the named graphs within it and associated with a `_default_` graph which is automatically assigned a *BNode* for an identifier - if one isn't given.

see *ConjunctiveGraph*

Quoted graph

The notion of an RDF graph [14] is extended to include the concept of a formula node. A formula node may occur wherever any other kind of node can appear. Associated with a formula node is an RDF graph that is completely disjoint from all other graphs; i.e. has no nodes in common with any other graph. (It may contain the same labels as other RDF graphs; because this is, by definition, a separate graph, considerations of tidiness do not apply between the graph at a formula node and any other graph.)

This is intended to map the idea of "{ N3-expression }" that is used by N3 into an RDF graph upon which RDF semantics is defined.

see *QuotedGraph*

Dataset

The RDF 1.1 Dataset, a small extension to the Conjunctive Graph. The primary term is "graphs in the datasets" and not "contexts with quads" so there is a separate method to set/retrieve a graph in a dataset and to operate with dataset graphs. As a consequence of this approach, dataset graphs cannot be identified with blank nodes, a name is always required (RDFLib will automatically add a name if one is not provided at creation time). This implementation includes a convenience method to directly add a single quad to a dataset graph.

see *Dataset*

Working with graphs

Instantiating Graphs with default store (Memory) and default identifier (a BNode):

```
>>> g = Graph()
>>> g.store.__class__
<class 'rdflib.plugins.stores.memory.Memory'>
>>> g.identifier.__class__
<class 'rdflib.term.BNode'>
```

Instantiating Graphs with a Memory store and an identifier - <https://rdflib.github.io>:

```
>>> g = Graph('Memory', URIRef("https://rdflib.github.io"))
>>> g.identifier
rdflib.term.URIRef('https://rdflib.github.io')
>>> str(g)
"<https://rdflib.github.io> a rdfg:Graph;rdflib:storage
 [a rdflib:Store;rdfs:label 'Memory']."
```

Creating a ConjunctiveGraph - The top level container for all named Graphs in a "database":

```
>>> g = ConjunctiveGraph()
>>> str(g.default_context)
"[a rdfg:Graph;rdflib:storage [a rdflib:Store;rdfs:label 'Memory']]."
```

Adding / removing reified triples to Graph and iterating over it directly or via triple pattern:

```
>>> g = Graph()
>>> statementId = BNode()
>>> print(len(g))
0
>>> g.add((statementId, RDF.type, RDF.Statement))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((statementId, RDF.subject,
...     URIRef("https://rdflib.github.io/store/ConjunctiveGraph")))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((statementId, RDF.predicate, namespace.RDFS.label))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((statementId, RDF.object, Literal("Conjunctive Graph")))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> print(len(g))
4
>>> for s, p, o in g:
```

(continues on next page)

(continued from previous page)

```
...     print(type(s))
...
<class 'rdflib.term.BNode'>
<class 'rdflib.term.BNode'>
<class 'rdflib.term.BNode'>
<class 'rdflib.term.BNode'>
```

```
>>> for s, p, o in g.triples((None, RDF.object, None)):
...     print(o)
...
Conjunctive Graph
>>> g.remove((statementId, RDF.type, RDF.Statement))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> print(len(g))
3
```

None terms in calls to *triples()* can be thought of as "open variables".

Graph support set-theoretic operators, you can add/subtract graphs, as well as intersection (with multiplication operator g1*g2) and xor (g1 ^ g2).

Note that BNode IDs are kept when doing set-theoretic operations, this may or may not be what you want. Two named graphs within the same application probably want share BNode IDs, two graphs with data from different sources probably not. If your BNode IDs are all generated by RDFLib they are UUIDs and unique.

```
>>> g1 = Graph()
>>> g2 = Graph()
>>> u = URIRef("http://example.com/foo")
>>> g1.add([u, namespace.RDFS.label, Literal("foo")])
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g1.add([u, namespace.RDFS.label, Literal("bar")])
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g2.add([u, namespace.RDFS.label, Literal("foo")])
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g2.add([u, namespace.RDFS.label, Literal("bing")])
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> len(g1 + g2)  # adds bing as label
3
>>> len(g1 - g2)  # removes foo
1
>>> len(g1 * g2)  # only foo
1
>>> g1 += g2  # now g1 contains everything
```

Graph Aggregation - ConjunctiveGraphs and ReadOnlyGraphAggregate within the same store:

```
>>> store = plugin.get("Memory", Store)()
>>> g1 = Graph(store)
>>> g2 = Graph(store)
>>> g3 = Graph(store)
>>> stmt1 = BNode()
>>> stmt2 = BNode()
>>> stmt3 = BNode()
```

(continues on next page)

(continued from previous page)

```
>>> g1.add((stmt1, RDF.type, RDF.Statement))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g1.add((stmt1, RDF.subject,
...     URIRef('https://rdflib.github.io/store/ConjunctiveGraph')))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g1.add((stmt1, RDF.predicate, namespace.RDFS.label))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g1.add((stmt1, RDF.object, Literal('Conjunctive Graph')))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g2.add((stmt2, RDF.type, RDF.Statement))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g2.add((stmt2, RDF.subject,
...     URIRef('https://rdflib.github.io/store/ConjunctiveGraph')))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g2.add((stmt2, RDF.predicate, RDF.type))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g2.add((stmt2, RDF.object, namespace.RDFS.Class))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g3.add((stmt3, RDF.type, RDF.Statement))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g3.add((stmt3, RDF.subject,
...     URIRef('https://rdflib.github.io/store/ConjunctiveGraph')))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g3.add((stmt3, RDF.predicate, namespace.RDFS.comment))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g3.add((stmt3, RDF.object, Literal(
...     'The top-level aggregate graph - The sum ' +
...     'of all named graphs within a Store')))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> len(list(ConjunctiveGraph(store).subjects(RDF.type, RDF.Statement)))
3
>>> len(list(ReadOnlyGraphAggregate([g1,g2]).subjects(
...     RDF.type, RDF.Statement)))
2
```

ConjunctiveGraphs have a *quads()* method which returns quads instead of triples, where the fourth item is the Graph (or subclass thereof) instance in which the triple was asserted:

```
>>> uniqueGraphNames = set(
...     [graph.identifier for s, p, o, graph in ConjunctiveGraph(store
...     ).quads((None, RDF.predicate, None))])
>>> len(uniqueGraphNames)
3
>>> unionGraph = ReadOnlyGraphAggregate([g1, g2])
>>> uniqueGraphNames = set(
...     [graph.identifier for s, p, o, graph in unionGraph.quads(
...     (None, RDF.predicate, None))])
>>> len(uniqueGraphNames)
2
```

Parsing N3 from a string

```
>>> g2 = Graph()
>>> src = '''
... @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
... @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
... [ a rdf:Statement ;
...   rdf:subject <https://rdflib.github.io/store#ConjunctiveGraph>;
...   rdf:predicate rdfs:label;
...   rdf:object "Conjunctive Graph" ] .
... '''
>>> g2 = g2.parse(data=src, format="n3")
>>> print(len(g2))
4
```

Using Namespace class:

```
>>> RDFLib = Namespace("https://rdflib.github.io/")
>>> RDFLib.ConjunctiveGraph
rdflib.term.URIRef('https://rdflib.github.io/ConjunctiveGraph')
>>> RDFLib["Graph"]
rdflib.term.URIRef('https://rdflib.github.io/Graph')
```

class rdflib.graph.**BatchAddGraph**(*graph*, *batch_size=1000*, *batch_addn=False*)

Bases: object

Wrapper around graph that turns batches of calls to Graph's add (and optionally, addN) into calls to batched calls to addN`.

Parameters

- graph: The graph to wrap
- batch_size: The maximum number of triples to buffer before passing to Graph's addN
- batch_addn: If True, then even calls to *addN* will be batched according to batch_size

graph: The wrapped graph count: The number of triples buffered since initialization or the last call to reset batch: The current buffer of triples

Parameters

- **graph** (*Graph*) –
- **batch_size** (int) –
- **batch_addn** (bool) –

__dict__ = mappingproxy({'__module__': 'rdflib.graph', '__doc__': "\n Wrapper around graph that turns batches of calls to Graph's add\n (and optionally, addN) into calls to batched calls to addN`.\n\n :Parameters:\n\n - graph: The graph to wrap\n - batch_size: The maximum number of triples to buffer before passing to\n Graph's addN\n - batch_addn: If True, then even calls to `addN` will be batched according to\n batch_size\n\n graph: The wrapped graph\n count: The number of triples buffered since initialization or the last call to reset\n batch: The current buffer of triples\n\n ", '__init__': <function BatchAddGraph.__init__>, 'reset': <function BatchAddGraph.reset>, 'add': <function BatchAddGraph.add>, 'addN': <function BatchAddGraph.addN>, '__enter__': <function BatchAddGraph.__enter__>, '__exit__': <function BatchAddGraph.__exit__>, '__dict__': <attribute '__dict__' of 'BatchAddGraph' objects>, '__weakref__': <attribute '__weakref__' of 'BatchAddGraph' objects>, '__annotations__': {}})

__enter__()

Return type
BatchAddGraph

__exit__(**exc*)

Return type
None

__init__(*graph*, *batch_size=1000*, *batch_addn=False*)

Parameters

- **graph** (*Graph*) –
- **batch_size** (int) –
- **batch_addn** (bool) –

__module__ = 'rdflib.graph'

__weakref__
list of weak references to the object (if defined)

add(*triple_or_quad*)
Add a triple to the buffer

Parameters

- **triple** – The triple to add
- **triple_or_quad** (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
BatchAddGraph

addN(*quads*)

Parameters
quads (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
BatchAddGraph

reset()
Manually clear the buffered triples and reset the count to zero

Return type
BatchAddGraph

class rdflib.graph.**ConjunctiveGraph**(*store='default'*, *identifier=None*, *default_graph_base=None*)
Bases: *Graph*

A ConjunctiveGraph is an (unnamed) aggregation of all the named graphs in a store.

It has a default graph, whose name is associated with the graph throughout its life. *__init__()* can take an identifier to use as the name of this default graph or it will assign a BNode.

All methods that add triples work against this default graph.

All queries are carried out against the union of all graphs.

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **default_graph_base** (Optional[str]) –

__contains__(*triple_or_quad*)

Support for 'triple/quad in graph' syntax

Parameters
triple_or_quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]]]) –

Return type
bool

__init__(*store='default'*, *identifier=None*, *default_graph_base=None*)

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **default_graph_base** (Optional[str]) –

__len__()

Number of triples in the entire conjunctive graph

Return type
int

__module__ = 'rdflib.graph'

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Graph*], Tuple[*Store*, *IdentifiedNode*]]

__str__()

Return str(self).

Return type
str

add(*triple_or_quad*)

Add a triple or quad to the store.

if a triple is given it is added to the default context

Parameters

- **self** (TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)) –
- **triple_or_quad** (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]]]) –

Return type
TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)

addN(*quads*)

Add a sequence of triples with context

Parameters

- **self** (TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)) –
- **quads** (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)

context_id(*uri*, *context_id=None*)

URI#context

Parameters

- **uri** (str) –
- **context_id** (Optional[str]) –

Return type
URIRef

contexts(*triple=None*)

Iterate over all contexts in the graph

If triple is specified, iterate over all contexts the triple is in.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*Graph*, None, None]

get_context(*identifier*, *quoted=False*, *base=None*)

Return a context graph for the given identifier

identifier must be a URIRef or BNode.

Parameters

- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **quoted** (bool) –
- **base** (Optional[str]) –

Return type
Graph

get_graph(*identifier*)

Returns the graph identified by given identifier

Parameters
identifier (*IdentifiedNode*) –

Return type
Optional[*Graph*]

parse(*source=None*, *publicID=None*, *format=None*, *location=None*, *file=None*, *data=None*, ***args*)

Parse source adding the resulting triples to its own context (sub graph of this graph).

See *rdflib.graph.Graph.parse()* for documentation on arguments.

Returns

The graph into which the source was parsed. In the case of n3 it returns the root context.

> **Caution:** This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.
>
> When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –
- **location** (Optional[str]) –
- **file** (Union[BinaryIO, TextIO, None]) –
- **data** (Union[str, bytes, None]) –
- **args** (Any) –

Return type
Graph

quads(*triple_or_quad=None*)

Iterate over all the quads in the entire conjunctive graph

Parameters
triple_or_quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]], None]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]], None, None]

remove(*triple_or_quad*)

Removes a triple or quads

if a triple is given it is removed from all contexts

a quad is removed from the given context only

Parameters

- **self** (TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)) –
- **triple_or_quad** (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]]]) –

Return type
TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)

remove_context(*context*)

Removes the given context from the graph

Parameters
context (*Graph*) –

Return type
None

triples(*triple_or_quad: Union[Tuple[Optional[Node], Optional[Node], Optional[Node]], Tuple[Optional[Node], Optional[Node], Optional[Node], Optional[Graph]]], context: Optional[Graph] = None*) → Generator[Tuple[*Node*, *Node*, *Node*], None, None]

triples(*triple_or_quad: Union[Tuple[Optional[Node], Path, Optional[Node]], Tuple[Optional[Node], Path, Optional[Node], Optional[Graph]]], context: Optional[Graph] = None*) → Generator[Tuple[*Node*, *Path*, *Node*], None, None]

triples(*triple_or_quad: Union[Tuple[Optional[Node], Optional[Union[Path, Node]], Optional[Node]], Tuple[Optional[Node], Optional[Union[Path, Node]], Optional[Node], Optional[Graph]]], context: Optional[Graph] = None*) → Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

Iterate over all the triples in the entire conjunctive graph

For legacy reasons, this can take the context to query either as a fourth element of the quad, or as the explicit context keyword parameter. The kw param takes precedence.

Parameters

- **triple_or_quad** (Union[Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]], Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*], Optional[*Graph*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

triples_choices(*triple*, *context=None*)

Iterate over all the triples in the entire conjunctive graph

Parameters

- **triple** (Union[Tuple[List[*Node*], *Node*, *Node*], Tuple[*Node*, List[*Node*], *Node*], Tuple[*Node*, *Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

class rdflib.graph.**Dataset**(*store='default'*, *default_union=False*, *default_graph_base=None*)

Bases: *ConjunctiveGraph*

RDF 1.1 Dataset. Small extension to the Conjunctive Graph: - the primary term is graphs in the datasets and not contexts with quads, so there is a separate method to set/retrieve a graph in a dataset and operate with graphs - graphs cannot be identified with blank nodes - added a method to directly add a single quad

Examples of usage:

```
>>> # Create a new Dataset
>>> ds = Dataset()
>>> # simple triples goes to default graph
>>> ds.add((URIRef("http://example.org/a"),
...    URIRef("http://www.example.org/b"),
```

(continues on next page)

(continued from previous page)

```
...    Literal("foo")))
<Graph identifier=... (<class 'rdflib.graph.Dataset'>)>
>>>
>>> # Create a graph in the dataset, if the graph name has already been
>>> # used, the corresponding graph will be returned
>>> # (ie, the Dataset keeps track of the constituent graphs)
>>> g = ds.graph(URIRef("http://www.example.com/gr"))
>>>
>>> # add triples to the new graph as usual
>>> g.add(
...     (URIRef("http://example.org/x"),
...     URIRef("http://example.org/y"),
...     Literal("bar")) )
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> # alternatively: add a quad to the dataset -> goes to the graph
>>> ds.add(
...     (URIRef("http://example.org/x"),
...     URIRef("http://example.org/z"),
...     Literal("foo-bar"),g) )
<Graph identifier=... (<class 'rdflib.graph.Dataset'>)>
>>>
>>> # querying triples return them all regardless of the graph
>>> for t in ds.triples((None,None,None)):
...     print(t)
(rdflib.term.URIRef("http://example.org/a"),
 rdflib.term.URIRef("http://www.example.org/b"),
 rdflib.term.Literal("foo"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"))
>>>
>>> # querying quads() return quads; the fourth argument can be unrestricted
>>> # (None) or restricted to a graph
>>> for q in ds.quads((None, None, None, None)):
...     print(q)
(rdflib.term.URIRef("http://example.org/a"),
 rdflib.term.URIRef("http://www.example.org/b"),
 rdflib.term.Literal("foo"),
 None)
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
>>>
>>> # unrestricted looping is equivalent to iterating over the entire Dataset
```

(continues on next page)

(continued from previous page)

```
>>> for q in ds:
...     print(q)
(rdflib.term.URIRef("http://example.org/a"),
 rdflib.term.URIRef("http://www.example.org/b"),
 rdflib.term.Literal("foo"),
 None)
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
>>>
>>> # resticting iteration to a graph:
>>> for q in ds.quads((None, None, None, g)):
...     print(q)
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
>>> # Note that in the call above -
>>> # ds.quads((None,None,None,"http://www.example.com/gr"))
>>> # would have been accepted, too
>>>
>>> # graph names in the dataset can be queried:
>>> for c in ds.graphs():
...     print(c)  # doctest:
DEFAULT
http://www.example.com/gr
>>> # A graph can be created without specifying a name; a skolemized genid
>>> # is created on the fly
>>> h = ds.graph()
>>> for c in ds.graphs():
...     print(c)
DEFAULT
https://rdflib.github.io/.well-known/genid/rdflib/N...
http://www.example.com/gr
>>> # Note that the Dataset.graphs() call returns names of empty graphs,
>>> # too. This can be restricted:
>>> for c in ds.graphs(empty=False):
...     print(c)
DEFAULT
http://www.example.com/gr
>>>
>>> # a graph can also be removed from a dataset via ds.remove_graph(g)
```

New in version 4.0.

Parameters

- **store** (Union[*Store*, str]) –
- **default_union** (bool) –
- **default_graph_base** (Optional[str]) –

__getstate__()

Return type
Tuple[*Store*, *IdentifiedNode*, *Graph*, bool]

__init__(*store='default'*, *default_union=False*, *default_graph_base=None*)

Parameters

- **store** (Union[*Store*, str]) –
- **default_union** (bool) –
- **default_graph_base** (Optional[str]) –

__iter__()

Iterates over all quads in the store

Return type
Generator[Tuple[*Node*, *Node*, *Node*, Optional[*IdentifiedNode*]], None, None]

__module__ = 'rdflib.graph'

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Dataset*], Tuple[*Store*, bool]]

__setstate__(*state*)

Parameters
state (Tuple[*Store*, *IdentifiedNode*, *Graph*, bool]) –

Return type
None

__str__()

Return str(self).

Return type
str

add_graph(*g*)

alias of graph for consistency

Parameters
g (Union[*IdentifiedNode*, *Graph*, str, None]) –

Return type
Graph

contexts(*triple=None*)

Iterate over all contexts in the graph

If triple is specified, iterate over all contexts the triple is in.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*Graph*, None, None]

default_context: *Graph*

graph(*identifier=None*, *base=None*)

Parameters

- **identifier** (Union[*IdentifiedNode*, *Graph*, str, None]) –
- **base** (Optional[str]) –

Return type
Graph

graphs(*triple=None*)

Iterate over all contexts in the graph

If triple is specified, iterate over all contexts the triple is in.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*Graph*, None, None]

parse(*source=None*, *publicID=None*, *format=None*, *location=None*, *file=None*, *data=None*, ***args*)

> **Caution:** This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.
>
> When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –
- **location** (Optional[str]) –
- **file** (Union[BinaryIO, TextIO, None]) –
- **data** (Union[str, bytes, None]) –
- **args** (Any) –

Return type
Graph

quads(*quad=None*)

Iterate over all the quads in the entire conjunctive graph

Parameters
quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]], None]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*, Optional[*IdentifiedNode*]], None, None]

remove_graph(*g*)

Parameters

- **self** (TypeVar(_DatasetT, bound= *Dataset*)) –
- **g** (Union[*IdentifiedNode*, *Graph*, str, None]) –

Return type
TypeVar(_DatasetT, bound= *Dataset*)

class rdflib.graph.**Graph**(*store='default'*, *identifier=None*, *namespace_manager=None*, *base=None*, *bind_namespaces='core'*)

Bases: *Node*

An RDF Graph

The constructor accepts one argument, the "store" that will be used to store the graph data (see the "store" package for stores currently shipped with rdflib).

Stores can be context-aware or unaware. Unaware stores take up (some) less space but cannot support features that require context, such as true merging/demerging of sub-graphs and provenance.

Even if used with a context-aware store, Graph will only expose the quads which belong to the default graph. To access the rest of the data, *ConjunctiveGraph* or *Dataset* classes can be used instead.

The Graph constructor can take an identifier which identifies the Graph by name. If none is given, the graph is assigned a BNode for its identifier.

For more on named graphs, see: http://www.w3.org/2004/03/trix/

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **namespace_manager** (Optional[*NamespaceManager*]) –
- **base** (Optional[str]) –
- **bind_namespaces** (Literal['core', 'rdflib', 'none']) –

__add__(*other*)

Set-theoretic union BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__and__(*other*)

Set-theoretic intersection. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__cmp__(*other*)

Return type
int

__contains__(*triple*)

Support for 'triple in graph' syntax

Parameters
triple (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type
bool

__dict__ = mappingproxy({'__module__': 'rdflib.graph', '__doc__': 'An RDF Graph\n\n The constructor accepts one argument, the "store"\n that will be used to store the graph data (see the "store"\n package for stores currently shipped with rdflib).\n\n Stores can be context-aware or unaware. Unaware stores take up\n (some) less space but cannot support features that require\n context, such as true merging/demerging of sub-graphs and\n provenance.\n\n Even if used with a context-aware store, Graph will only expose the quads which\n belong to the default graph. To access the rest of the data, `ConjunctiveGraph` or\n `Dataset` classes can be used instead.\n\n The Graph constructor can take an identifier which identifies the Graph\n by name. If none is given, the graph is assigned a BNode for its\n identifier.\n\n For more on named graphs, see: http://www.w3.org/2004/03/trix/\n ', '__init__': <function Graph.__init__>, 'store': <property object>, 'identifier': <property object>, 'namespace_manager': <property object>, '__repr__': <function Graph.__repr__>, '__str__': <function Graph.__str__>, 'toPython': <function Graph.toPython>, 'destroy': <function Graph.destroy>, 'commit': <function Graph.commit>, 'rollback': <function Graph.rollback>, 'open': <function Graph.open>, 'close': <function Graph.close>, 'add': <function Graph.add>, 'addN': <function Graph.addN>, 'remove': <function Graph.remove>, 'triples': <function Graph.triples>, '__getitem__': <function Graph.__getitem__>, '__len__': <function Graph.__len__>, '__iter__': <function Graph.__iter__>, '__contains__': <function Graph.__contains__>, '__hash__': <function Graph.__hash__>, '__cmp__': <function Graph.__cmp__>, '__eq__': <function Graph.__eq__>, '__lt__': <function Graph.__lt__>, '__le__': <function Graph.__le__>, '__gt__': <function Graph.__gt__>, '__ge__': <function Graph.__ge__>, '__iadd__': <function Graph.__iadd__>, '__isub__': <function Graph.__isub__>, '__add__': <function Graph.__add__>, '__mul__': <function Graph.__mul__>, '__sub__': <function Graph.__sub__>, '__xor__': <function Graph.__xor__>, '__or__': <function Graph.__add__>, '__and__': <function Graph.__mul__>, 'set': <function Graph.set>, 'subjects': <function Graph.subjects>, 'predicates': <function Graph.predicates>, 'objects': <function Graph.objects>, 'subject_predicates': <function Graph.subject_predicates>, 'subject_objects': <function Graph.subject_objects>, 'predicate_objects': <function Graph.predicate_objects>, 'triples_choices': <function Graph.triples_choices>, 'value': <function Graph.value>, 'items': <function Graph.items>, 'transitiveClosure': <function Graph.transitiveClosure>, 'transitive_objects': <function Graph.transitive_objects>, 'transitive_subjects': <function Graph.transitive_subjects>, 'qname': <function Graph.qname>, 'compute_qname': <function Graph.compute_qname>, 'bind': <function Graph.bind>, 'namespaces': <function Graph.namespaces>, 'absolutize': <function Graph.absolutize>, 'serialize': <function Graph.serialize>, 'print': <function Graph.print>, 'parse': <function Graph.parse>, 'query': <function Graph.query>, 'update': <function Graph.update>, 'n3': <function Graph.n3>, '__reduce__': <function Graph.__reduce__>, 'isomorphic': <function Graph.isomorphic>, 'connected': <function Graph.connected>, 'all_nodes': <function Graph.all_nodes>, 'collection': <function Graph.collection>, 'resource': <function Graph.resource>, '_process_skolem_tuples': <function Graph._process_skolem_tuples>, 'skolemize': <function Graph.skolemize>, 'de_skolemize': <function Graph.de_skolemize>, 'cbd': <function Graph.cbd>, '__dict__': <attribute '__dict__' of 'Graph' objects>, '__weakref__': <attribute '__weakref__' of 'Graph' objects>, '__annotations__': {'__identifier': '_ContextIdentifierType', '__store': 'Store'}})

__eq__(*other*)

Return self==value.

Return type

bool

__ge__(*other*)

Return self>=value.

Parameters
other (*Graph*) –

Return type
bool

__getitem__(*item*)

A graph can be "sliced" as a shortcut for the triples method The python slice syntax is (ab)used for specifying triples. A generator over matches is returned, the returned tuples include only the parts not given

```
>>> import rdflib
>>> g = rdflib.Graph()
>>> g.add((rdflib.URIRef("urn:bob"), namespace.RDFS.label, rdflib.Literal("Bob
↪")))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
```

```
>>> list(g[rdflib.URIRef("urn:bob")]) # all triples about bob
[(rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#label'), rdflib.term.
↪Literal('Bob'))]
```

```
>>> list(g[:namespace.RDFS.label]) # all label triples
[(rdflib.term.URIRef('urn:bob'), rdflib.term.Literal('Bob'))]
```

```
>>> list(g[::rdflib.Literal("Bob")]) # all triples with bob as object
[(rdflib.term.URIRef('urn:bob'), rdflib.term.URIRef('http://www.w3.org/2000/01/
↪rdf-schema#label'))]
```

Combined with SPARQL paths, more complex queries can be written concisely:

Name of all Bobs friends:

g[bob : FOAF.knows/FOAF.name]

Some label for Bob:

g[bob : DC.title|FOAF.name|RDFS.label]

All friends and friends of friends of Bob

g[bob : FOAF.knows * "+"]

etc.

New in version 4.0.

__gt__(*other*)

Return self>value.

Return type
bool

__hash__()

Return hash(self).

Return type
int

__iadd__(*other*)

Add all triples in Graph other to Graph. BNode IDs are not changed.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **other** (Iterable[Tuple[*Node*, *Node*, *Node*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

__init__(*store='default'*, *identifier=None*, *namespace_manager=None*, *base=None*, *bind_namespaces='core'*)

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **namespace_manager** (Optional[*NamespaceManager*]) –
- **base** (Optional[str]) –
- **bind_namespaces** (Literal['core', 'rdflib', 'none']) –

__isub__(*other*)

Subtract all triples in Graph other from Graph. BNode IDs are not changed.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **other** (Iterable[Tuple[*Node*, *Node*, *Node*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

__iter__()

Iterates over all triples in the store

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

__le__(*other*)

Return self<=value.

Parameters
other (*Graph*) –

Return type
bool

__len__()

Returns the number of triples in the graph

If context is specified then the number of triples in the context is returned instead.

Return type
int

__lt__(*other*)

Return self<value.

Return type
bool

__module__ = 'rdflib.graph'

__mul__(*other*)

Set-theoretic intersection. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__or__(*other*)

Set-theoretic union BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Graph*], Tuple[*Store*, *IdentifiedNode*]]

__repr__()

Return repr(self).

Return type
str

__str__()

Return str(self).

Return type
str

__sub__(*other*)

Set-theoretic difference. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__weakref__

list of weak references to the object (if defined)

__xor__(*other*)

Set-theoretic XOR. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

absolutize(*uri*, *defrag=1*)

Turn uri into an absolute URI if it's not one already

Parameters

- **uri** (str) –
- **defrag** (int) –

Return type
URIRef

add(*triple*)

Add a triple with self as context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[*Node*, *Node*, *Node*]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

addN(*quads*)

Add a sequence of triple with context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **quads** (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

all_nodes()

Return type
Set[*Node*]

bind(*prefix*, *namespace*, *override=True*, *replace=False*)

Bind prefix to namespace

If override is True will bind namespace to given prefix even if namespace was already bound to a different prefix.

if replace, replace any existing prefix with the new namespace

for example: graph.bind("foaf", "http://xmlns.com/foaf/0.1/")

Parameters

- **prefix** (Optional[str]) –
- **namespace** (Any) –
- **override** (bool) –
- **replace** (bool) –

Return type
None

cbd(*resource*)

Retrieves the Concise Bounded Description of a Resource from a Graph

Concise Bounded Description (CBD) is defined in [1] as:

Given a particular node (the starting node) in a particular RDF graph (the source graph), a subgraph of that particular graph, taken to comprise a concise bounded description of the resource denoted by the starting node, can be identified as follows:

1. **Include in the subgraph all statements in the source graph where the subject of the statement is the**
   starting node;
2. **Recursively, for all statements identified in the subgraph thus far having a blank node object, include**
   in the subgraph all statements in the source graph where the subject of the statement is the blank node in question and which are not already included in the subgraph.
3. **Recursively, for all statements included in the subgraph thus far, for all reifications of each statement**
   in the source graph, include the concise bounded description beginning from the rdf:Statement node of each reification.

This results in a subgraph where the object nodes are either URI references, literals, or blank nodes not serving as the subject of any statement in the graph.

[1] https://www.w3.org/Submission/CBD/

Parameters
resource (*Node*) – a URIRef object, of the Resource for queried for

Return type
Graph

Returns
a Graph, subgraph of self

close(*commit_pending_transaction=False*)

Close the graph store

Might be necessary for stores that require closing a connection to a database or releasing some resource.

Parameters
commit_pending_transaction (bool) –

Return type
None

collection(*identifier*)

Create a new `Collection` instance.

Parameters:

- `identifier`: a URIRef or BNode instance.

Example:

```
>>> graph = Graph()
>>> uri = URIRef("http://example.org/resource")
>>> collection = graph.collection(uri)
>>> assert isinstance(collection, Collection)
>>> assert collection.uri is uri
```

(continues on next page)

(continued from previous page)

```
>>> assert collection.graph is graph
>>> collection += [ Literal(1), Literal(2) ]
```

Parameters
identifier (*Node*) –

Return type
Collection

commit()

Commits active transactions

Parameters
self (TypeVar(_GraphT, bound= *Graph*)) –

Return type
TypeVar(_GraphT, bound= *Graph*)

compute_qname(*uri*, *generate=True*)

Parameters

- **uri** (str) –
- **generate** (bool) –

Return type
Tuple[str, *URIRef*, str]

connected()

Check if the Graph is connected

The Graph is considered undirectional.

Performs a search on the Graph, starting from a random node. Then iteratively goes depth-first through the triplets where the node is subject and object. Return True if all nodes have been visited and False if it cannot continue and there are still unvisited nodes left.

Return type
bool

de_skolemize(*new_graph=None*, *uriref=None*)

Parameters

- **new_graph** (Optional[*Graph*]) –
- **uriref** (Optional[*URIRef*]) –

Return type
Graph

destroy(*configuration*)

Destroy the store identified by `configuration` if supported

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **configuration** (str) –

Return type
TypeVar(_GraphT, bound= *Graph*)

property identifier: *IdentifiedNode*

Return type
IdentifiedNode

isomorphic(*other*)

does a very basic check if these graphs are the same If no BNodes are involved, this is accurate.

See rdflib.compare for a correct implementation of isomorphism checks

Parameters
other (*Graph*) –

Return type
bool

items(*list*)

Generator over all items in the resource specified by list

list is an RDF collection.

Parameters
list (*Node*) –

Return type
Generator[*Node*, None, None]

n3()

Return an n3 identifier for the Graph

Return type
str

property namespace_manager: *NamespaceManager*

this graph's namespace-manager

Return type
NamespaceManager

namespaces()

Generator over all the prefix, namespace tuples

Return type
Generator[Tuple[str, *URIRef*], None, None]

objects(*subject=None*, *predicate=None*, *unique=False*)

A generator of (optionally unique) objects with the given subject and predicate

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Union[None, *Path*, *Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

open(*configuration*, *create=False*)

Open the graph store

Might be necessary for stores that require opening a connection to a database or acquiring some resource.

Parameters

- **configuration** (str) –
- **create** (bool) –

Return type
Optional[int]

parse(*source=None*, *publicID=None*, *format=None*, *location=None*, *file=None*, *data=None*, ***args*)

Parse an RDF source adding the resulting triples to the Graph.

The source is specified using one of source, location, file or data.

> **Caution:** This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.
>
> When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- `source`: An InputSource, file-like object, or string. In the case of a string the string is the location of the source.
- `location`: A string indicating the relative or absolute URL of the source. Graph's absolutize method is used if a relative location is specified.
- `file`: A file-like object.
- `data`: A string containing the data to be parsed.
- `format`: Used if format can not be determined from source, e.g. file extension or Media Type. Defaults to text/turtle. Format support can be extended with plugins, but "xml", "n3" (use for turtle), "nt" & "trix" are built in.
- `publicID`: the logical URI to use as the document base. If None specified the document location is used (at least in the case where there is a document location).

Returns

- self, the graph instance.

Examples:

```
>>> my_data = '''
... <rdf:RDF
...   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
...   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
... >
...   <rdf:Description>
...     <rdfs:label>Example</rdfs:label>
```

(continues on next page)

(continued from previous page)

```
...     <rdfs:comment>This is really just an example.</rdfs:comment>
...   </rdf:Description>
... </rdf:RDF>
... '''
>>> import os, tempfile
>>> fd, file_name = tempfile.mkstemp()
>>> f = os.fdopen(fd, "w")
>>> dummy = f.write(my_data)  # Returns num bytes written
>>> f.close()
```

```
>>> g = Graph()
>>> result = g.parse(data=my_data, format="application/rdf+xml")
>>> len(g)
2
```

```
>>> g = Graph()
>>> result = g.parse(location=file_name, format="application/rdf+xml")
>>> len(g)
2
```

```
>>> g = Graph()
>>> with open(file_name, "r") as f:
...     result = g.parse(f, format="application/rdf+xml")
>>> len(g)
2
```

```
>>> os.remove(file_name)
```

```
>>> # default turtle parsing
>>> result = g.parse(data="<http://example.com/a> <http://example.com/a> <http:/
↪/example.com/a> .")
>>> len(g)
3
```

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –
- **location** (Optional[str]) –
- **file** (Union[BinaryIO, TextIO, None]) –
- **data** (Union[str, bytes, None]) –
- **args** (Any) –

Return type

Graph

predicate_objects(*subject=None*, *unique=False*)

A generator of (optionally unique) (predicate, object) tuples for the given subject

Parameters

- **subject** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

predicates(*subject=None*, *object=None*, *unique=False*)

A generator of (optionally unique) predicates with the given subject and object

Parameters

- **subject** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

print(*format='turtle'*, *encoding='utf-8'*, *out=None*)

Parameters

- **format** (str) –
- **encoding** (str) –
- **out** (Optional[TextIO]) –

Return type
None

qname(*uri*)

Parameters
uri (str) –

Return type
str

query(*query_object*, *processor='sparql'*, *result='sparql'*, *initNs=None*, *initBindings=None*, *use_store_provided=True*, ***kwargs*)

Query this graph.

A type of 'prepared queries' can be realised by providing initial variable bindings with initBindings

Initial namespaces are used to resolve prefixes used in the query, if none are given, the namespaces from the graph's namespace manager are used.

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in SERVICE directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Returntype
Result

Parameters

- **query_object** (Union[str, *Query*]) –
- **processor** (Union[str, *Processor*]) –
- **result** (Union[str, Type[*Result*]]) –
- **initNs** (Optional[Mapping[str, Any]]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –
- **use_store_provided** (bool) –
- **kwargs** (Any) –

Return type
Result

remove(*triple*)

Remove a triple from the graph

If the triple does not provide a context attribute, removes the triple from all contexts.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

resource(*identifier*)

Create a new `Resource` instance.

Parameters:

- `identifier`: a URIRef or BNode instance.

Example:

```
>>> graph = Graph()
>>> uri = URIRef("http://example.org/resource")
>>> resource = graph.resource(uri)
>>> assert isinstance(resource, Resource)
>>> assert resource.identifier is uri
>>> assert resource.graph is graph
```

Parameters
identifier (Union[*Node*, str]) –

Return type
Resource

rollback()

Rollback active transactions

Parameters
self (TypeVar(_GraphT, bound= *Graph*)) –

Return type
TypeVar(_GraphT, bound= *Graph*)

serialize(*destination: None*, *format: str*, *base: Optional[str]*, *encoding: str*, ***args: Any*) → bytes

serialize(*destination: None = None*, *format: str = 'turtle'*, *base: Optional[str] = None*, ***, *encoding: str*, ***args: Any*) → bytes

serialize(*destination: None = None*, *format: str = 'turtle'*, *base: Optional[str] = None*, *encoding: None = None*, ***args: Any*) → str

serialize(*destination: Union[str, PurePath, IO[bytes]]*, *format: str = 'turtle'*, *base: Optional[str] = None*, *encoding: Optional[str] = None*, ***args: Any*) → *Graph*

serialize(*destination: Optional[Union[str, PurePath, IO[bytes]]] = None*, *format: str = 'turtle'*, *base: Optional[str] = None*, *encoding: Optional[str] = None*, ***args: Any*) → Union[bytes, str, *Graph*]

Serialize the graph.

Parameters

- **destination** (Union[str, PurePath, IO[bytes], None]) – The destination to serialize the graph to. This can be a path as a str or PurePath object, or it can be a IO[bytes] like object. If this parameter is not supplied the serialized graph will be returned.
- **format** (str) – The format that the output should be written in. This value references a *Serializer* plugin. Format support can be extended with plugins, but "xml", "n3", "turtle", "nt", "pretty-xml", "trix", "trig", "nquads", "json-ld" and "hext" are built in. Defaults to "turtle".
- **base** (Optional[str]) – The base IRI for formats that support it. For the turtle format this will be used as the @base directive.
- **encoding** (Optional[str]) – Encoding of output.
- **args** (Any) – Additional arguments to pass to the *Serializer* that will be used.
- **self** (TypeVar(_GraphT, bound= *Graph*)) –

Returns
The serialized graph if destination is None. The serialized graph is returned as str if no encoding is specified, and as bytes if an encoding is specified.

Return type
bytes if destination is None and encoding is not None.

Return type
str if destination is None and encoding is None.

Returns
self (i.e. the *Graph* instance) if destination is not None.

Return type
Graph if destination is not None.

set(*triple*)

Convenience method to update the value of object

Remove any existing triples for subject and predicate before adding (subject, predicate, object).

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[*Node*, *Node*, *Node*]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

skolemize(*new_graph=None*, *bnode=None*, *authority=None*, *basepath=None*)

Parameters

- **new_graph** (Optional[*Graph*]) –
- **bnode** (Optional[*BNode*]) –
- **authority** (Optional[str]) –
- **basepath** (Optional[str]) –

Return type
Graph

property store: *Store*

Return type
Store

subject_objects(*predicate=None*, *unique=False*)

A generator of (optionally unique) (subject, object) tuples for the given predicate

Parameters

- **predicate** (Union[None, *Path*, *Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subject_predicates(*object=None*, *unique=False*)

A generator of (optionally unique) (subject, predicate) tuples for the given object

Parameters

- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subjects(*predicate=None*, *object=None*, *unique=False*)

A generator of (optionally unique) subjects with the given predicate and object

Parameters

- **predicate** (Union[None, *Path*, *Node*]) –
- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

toPython()

Parameters
self (TypeVar(_GraphT, bound= *Graph*)) –

Return type
TypeVar(_GraphT, bound= *Graph*)

transitiveClosure(*func*, *arg*, *seen=None*)

Generates transitive closure of a user-defined function against the graph

```
>>> from rdflib.collection import Collection
>>> g=Graph()
>>> a=BNode("foo")
>>> b=BNode("bar")
>>> c=BNode("baz")
>>> g.add((a,RDF.first,RDF.type))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((a,RDF.rest,b))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((b,RDF.first,namespace.RDFS.label))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((b,RDF.rest,c))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((c,RDF.first,namespace.RDFS.comment))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((c,RDF.rest,RDF.nil))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> def topList(node,g):
...    for s in g.subjects(RDF.rest, node):
...       yield s
>>> def reverseList(node,g):
...    for f in g.objects(node, RDF.first):
...       print(f)
...    for s in g.subjects(RDF.rest, node):
...       yield s
```

```
>>> [rt for rt in g.transitiveClosure(
...     topList,RDF.nil)]
[rdflib.term.BNode('baz'),
 rdflib.term.BNode('bar'),
 rdflib.term.BNode('foo')]
```

```
>>> [rt for rt in g.transitiveClosure(
...     reverseList,RDF.nil)]
http://www.w3.org/2000/01/rdf-schema#comment
http://www.w3.org/2000/01/rdf-schema#label
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
[rdflib.term.BNode('baz'),
 rdflib.term.BNode('bar'),
 rdflib.term.BNode('foo')]
```

Parameters

- **func** (Callable[[TypeVar(_TCArgT), *Graph*], Iterable[TypeVar(_TCArgT)]]) –
- **arg** (TypeVar(_TCArgT)) –
- **seen** (Optional[Dict[TypeVar(_TCArgT), int]]) –

transitive_objects(*subject*, *predicate*, *remember=None*)

Transitively generate objects for the `predicate` relationship

Generated objects belong to the depth first transitive closure of the `predicate` relationship starting at `subject`.

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –
- **remember** (Optional[Dict[Optional[*Node*], int]]) –

Return type
Generator[Optional[*Node*], None, None]

transitive_subjects(*predicate*, *object*, *remember=None*)

Transitively generate subjects for the `predicate` relationship

Generated subjects belong to the depth first transitive closure of the `predicate` relationship starting at `object`.

Parameters

- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –
- **remember** (Optional[Dict[Optional[*Node*], int]]) –

Return type
Generator[Optional[*Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]*) → Generator[Tuple[*Node*, *Node*, *Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Path, Optional[*Node*]]*) → Generator[Tuple[*Node*, *Path*, *Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Optional[Union[Path, *Node*]], Optional[*Node*]]*) → Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

Generator over the triple store

Returns triples that match the given triple pattern. If triple pattern does not provide a context, all contexts will be searched.

Parameters
triple (Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]]) –

Return type
Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

triples_choices(*triple*, *context=None*)

Parameters

- **triple** (Union[Tuple[List[*Node*], *Node*, *Node*], Tuple[*Node*, List[*Node*], *Node*], Tuple[*Node*, *Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

update(*update_object*, *processor='sparql'*, *initNs=None*, *initBindings=None*, *use_store_provided=True*, ***kwargs*)

Update this graph with the given update query.

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in `SERVICE` directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **update_object** (Union[*Update*, str]) –
- **processor** (Union[str, *UpdateProcessor*]) –
- **initNs** (Optional[Mapping[str, Any]]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –
- **use_store_provided** (bool) –
- **kwargs** (Any) –

Return type
None

value(*subject: None = None*, *predicate: None = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: Optional[Node] = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

value(*subject: Optional[Node] = None*, *predicate: None = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: None = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

value(*subject: None = None*, *predicate: Optional[Node] = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: None = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

value(*subject: Optional[Node] = None*, *predicate: Optional[Node] = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: Optional[Node] = None*, *default: Optional[Node] = None*, *any: bool = True*) → Optional[*Node*]

Get a value for a pair of two criteria

Exactly one of subject, predicate, object must be None. Useful if one knows that there may only be one value.

It is one of those situations that occur a lot, hence this 'macro' like utility

Parameters: subject, predicate, object – exactly one must be None default – value to be returned if no values found any – if True, return any value in the case there is more than one, else, raise UniquenessError

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –

- **default** (Optional[*Node*]) –
- **any** (bool) –

Return type
Optional[*Node*]

exception rdflib.graph.**ModificationException**

Bases: Exception

__init__()

__module__ = 'rdflib.graph'

__str__()

Return str(self).

Return type
str

__weakref__

list of weak references to the object (if defined)

class rdflib.graph.**QuotedGraph**(*store*, *identifier*)

Bases: *Graph*

Quoted Graphs are intended to implement Notation 3 formulae. They are associated with a required identifier that the N3 parser *must* provide in order to maintain consistent formulae identification for scenarios such as implication and other such processing.

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –

__init__(*store*, *identifier*)

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –

__module__ = 'rdflib.graph'

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Graph*], Tuple[*Store*, *IdentifiedNode*]]

__str__()

Return str(self).

Return type
str

add(*triple*)

Add a triple with self as context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –

- **triple** (Tuple[*Node*, *Node*, *Node*]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

addN(*quads*)

Add a sequence of triple with context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **quads** (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

n3()

Return an n3 identifier for the Graph

Return type
str

class rdflib.graph.**ReadOnlyGraphAggregate**(*graphs*, *store='default'*)

Bases: *ConjunctiveGraph*

Utility class for treating a set of graphs as a single graph

Only read operations are supported (hence the name). Essentially a ConjunctiveGraph over an explicit subset of the entire store.

Parameters

- **graphs** (List[*Graph*]) –
- **store** (Union[str, *Store*]) –

__cmp__(*other*)

Return type
int

__contains__(*triple_or_quad*)

Support for 'triple/quad in graph' syntax

Parameters
triple_or_quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]]]) –

Return type
bool

__hash__()

Return hash(self).

Return type
NoReturn

__iadd__(*other*)

Add all triples in Graph other to Graph. BNode IDs are not changed.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –

- **other** (Iterable[Tuple[*Node*, *Node*, *Node*]]) –

Return type
NoReturn

__init__(*graphs*, *store='default'*)

Parameters

- **graphs** (List[*Graph*]) –
- **store** (Union[str, *Store*]) –

__isub__(*other*)

Subtract all triples in Graph other from Graph. BNode IDs are not changed.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **other** (Iterable[Tuple[*Node*, *Node*, *Node*]]) –

Return type
NoReturn

__len__()

Number of triples in the entire conjunctive graph

Return type
int

__module__ = 'rdflib.graph'

__reduce__()

Helper for pickle.

Return type
NoReturn

__repr__()

Return repr(self).

Return type
str

absolutize(*uri*, *defrag=1*)

Turn uri into an absolute URI if it's not one already

Parameters

- **uri** (str) –
- **defrag** (int) –

Return type
NoReturn

add(*triple*)

Add a triple or quad to the store.

if a triple is given it is added to the default context

Parameters
triple (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]]]) –

Return type
NoReturn

addN(*quads*)

Add a sequence of triples with context

Parameters
quads (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
NoReturn

bind(*prefix*, *namespace*, *override=True*)

Bind prefix to namespace

If override is True will bind namespace to given prefix even if namespace was already bound to a different prefix.

if replace, replace any existing prefix with the new namespace

for example: graph.bind("foaf", "http://xmlns.com/foaf/0.1/")

Parameters

- **prefix** (Optional[str]) –
- **namespace** (Any) –
- **override** (bool) –

Return type
NoReturn

close()

Close the graph store

Might be necessary for stores that require closing a connection to a database or releasing some resource.

Return type
None

commit()

Commits active transactions

Return type
NoReturn

compute_qname(*uri*, *generate=True*)

Parameters

- **uri** (str) –
- **generate** (bool) –

Return type
Tuple[str, *URIRef*, str]

default_context: *Graph*

destroy(*configuration*)

Destroy the store identified by `configuration` if supported

Parameters
configuration (str) –

Return type
NoReturn

n3()

Return an n3 identifier for the Graph

Return type
NoReturn

namespaces()

Generator over all the prefix, namespace tuples

Return type
Generator[Tuple[str, *URIRef*], None, None]

open(*configuration*, *create=False*)

Open the graph store

Might be necessary for stores that require opening a connection to a database or acquiring some resource.

Parameters

- **configuration** (str) –
- **create** (bool) –

Return type
None

parse(*source*, *publicID=None*, *format=None*, ***args*)

Parse source adding the resulting triples to its own context (sub graph of this graph).

See *rdflib.graph.Graph.parse()* for documentation on arguments.

Returns

The graph into which the source was parsed. In the case of n3 it returns the root context.

Caution: This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.

When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.

For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –
- **args** (Any) –

Return type
NoReturn

qname(*uri*)

Parameters
uri (str) –

Return type
str

quads(*triple_or_quad*)

Iterate over all the quads in the entire aggregate graph

Parameters
triple_or_quad (Union[Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]], Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*], Optional[*Graph*]]]) –

Return type
Generator[Tuple[*Node*, Union[*Path*, *Node*], *Node*, *Graph*], None, None]

remove(*triple*)

Removes a triple or quads

if a triple is given it is removed from all contexts

a quad is removed from the given context only

Parameters
triple (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]]]) –

Return type
NoReturn

rollback()

Rollback active transactions

Return type
NoReturn

triples(*triple: Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]*) → Generator[Tuple[*Node*, *Node*, *Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Path, Optional[*Node*]]*) → Generator[Tuple[*Node*, *Path*, *Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Optional[Union[Path, *Node*]], Optional[*Node*]]*) → Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

Iterate over all the triples in the entire conjunctive graph

For legacy reasons, this can take the context to query either as a fourth element of the quad, or as the explicit context keyword parameter. The kw param takes precedence.

Parameters
triple (Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]]) –

Return type
Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

triples_choices(*triple*, *context=None*)

Iterate over all the triples in the entire conjunctive graph

Parameters

- **triple** (Union[Tuple[List[*Node*], *Node*, *Node*], Tuple[*Node*, List[*Node*], *Node*], Tuple[*Node*, *Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

class rdflib.graph.**Seq**(*graph*, *subject*)

Bases: object

Wrapper around an RDF Seq resource

It implements a container type in Python with the order of the items returned corresponding to the Seq content. It is based on the natural ordering of the predicate names _1, _2, _3, etc, which is the 'implementation' of a sequence in RDF terms.

Parameters

- **graph** (*Graph*) –
- **subject** (*Node*) –

__dict__ = mappingproxy({'__module__': 'rdflib.graph', '__doc__': "Wrapper around an RDF Seq resource\n\n It implements a container type in Python with the order of the items\n returned corresponding to the Seq content. It is based on the natural\n ordering of the predicate names _1, _2, _3, etc, which is the\n 'implementation' of a sequence in RDF terms.\n ", '__init__': <function Seq.__init__>, 'toPython': <function Seq.toPython>, '__iter__': <function Seq.__iter__>, '__len__': <function Seq.__len__>, '__getitem__': <function Seq.__getitem__>, '__dict__': <attribute '__dict__' of 'Seq' objects>, '__weakref__': <attribute '__weakref__' of 'Seq' objects>, '__annotations__': {'_list': 'List[Tuple[int, _ObjectType]]'}})

__getitem__(*index*)

Item given by index from the Seq

Return type
Node

__init__(*graph*, *subject*)

Parameters:

- **graph:**
  the graph containing the Seq
- **subject:**
  the subject of a Seq. Note that the init does not check whether this is a Seq, this is done in whoever creates this instance!

Parameters

- **graph** (*Graph*) –
- **subject** (*Node*) –

__iter__()

Generator over the items in the Seq

Return type
Generator[*Node*, None, None]

__len__()

Length of the Seq

Return type
int

__module__ = 'rdflib.graph'

__weakref__

list of weak references to the object (if defined)

toPython()

Return type
Seq

exception rdflib.graph.**UnSupportedAggregateOperation**

Bases: Exception

__init__()

__module__ = 'rdflib.graph'

__str__()

Return str(self).

Return type
str

__weakref__

list of weak references to the object (if defined)

rdflib.parser module

Parser plugin interface.

This module defines the parser plugin interface and contains other related parser support code.

The module is mainly useful for those wanting to write a parser that can plugin to rdflib. If you are wanting to invoke a parser you likely want to do so through the Graph class parse method.

class rdflib.parser.**FileInputSource**(*file*)

Bases: *InputSource*

Parameters
file (Union[BinaryIO, TextIO, TextIOBase, RawIOBase, BufferedIOBase]) –

__init__(*file*)

Parameters
file (Union[BinaryIO, TextIO, TextIOBase, RawIOBase, BufferedIOBase]) –

__module__ = 'rdflib.parser'

__repr__()

Return repr(self).

Return type
str

class rdflib.parser.**InputSource**(*system_id=None*)

Bases: InputSource

TODO:

Parameters
system_id (Optional[str]) –

__init__(*system_id=None*)

Parameters
system_id (Optional[str]) –

__module__ = 'rdflib.parser'

close()

Return type
None

class rdflib.parser.**Parser**

Bases: object

__init__()

__module__ = 'rdflib.parser'

__slots__ = ()

parse(*source*, *sink*)

Parameters

- **source** (*InputSource*) –
- **sink** (*Graph*) –

Return type
None

class rdflib.parser.**PythonInputSource**(*data*, *system_id=None*)

Bases: *InputSource*

Constructs an RDFLib Parser InputSource from a Python data structure, for example, loaded from JSON with json.load or json.loads:

```
>>> import json
>>> as_string = """{
...   "@context" : {"ex" : "http://example.com/ns#"},
...   "@graph": [{"@type": "ex:item", "@id": "#example"}]
... }"""
>>> as_python = json.loads(as_string)
>>> source = create_input_source(data=as_python)
>>> isinstance(source, PythonInputSource)
True
```

Parameters

- **data** (Any) –
- **system_id** (Optional[str]) –

__init__(*data*, *system_id=None*)

Parameters

- **data** (Any) –
- **system_id** (Optional[str]) –

__module__ = 'rdflib.parser'

close()

Return type
None

content_type: Optional[str]

getPublicId()

Returns the public identifier of this InputSource.

Return type
Optional[str]

getSystemId()

Returns the system identifier of this InputSource.

Return type
Optional[str]

setPublicId(*public_id*)

Sets the public identifier of this InputSource.

Parameters
public_id (Optional[str]) –

Return type
None

setSystemId(*system_id*)

Sets the system identifier of this InputSource.

Parameters
system_id (Optional[str]) –

Return type
None

class rdflib.parser.**StringInputSource**(*value*, *encoding='utf-8'*, *system_id=None*)

Bases: *InputSource*

Constructs an RDFLib Parser InputSource from a Python String or Bytes

Parameters

- **value** (Union[str, bytes]) –
- **encoding** (str) –
- **system_id** (Optional[str]) –

__init__(*value*, *encoding='utf-8'*, *system_id=None*)

Parameters

- **value** (Union[str, bytes]) –

- **encoding** (str) –
- **system_id** (Optional[str]) –

__module__ = 'rdflib.parser'

content_type: Optional[str]

class rdflib.parser.**URLInputSource**(*system_id=None*, *format=None*)

Bases: *InputSource*

Constructs an RDFLib Parser InputSource from a URL to read it from the Web.

Parameters

- **system_id** (Optional[str]) –
- **format** (Optional[str]) –

__annotations__ = {'links': 'List[str]'}

__init__(*system_id=None*, *format=None*)

Parameters

- **system_id** (Optional[str]) –
- **format** (Optional[str]) –

__module__ = 'rdflib.parser'

__repr__()

Return repr(self).

Return type
str

get_alternates(*type_=None*)

Parameters
type_ (Optional[str]) –

Return type
List[str]

classmethod **get_links**(*response*)

Parameters
response (addinfourl) –

Return type
List[str]

classmethod **getallmatchingheaders**(*message*, *name*)

Parameters
message (Message) –

Return type
List[str]

links: List[str]

rdflib.paths module

This module implements the SPARQL 1.1 Property path operators, as defined in:

http://www.w3.org/TR/sparql11-query/#propertypaths

In SPARQL the syntax is as follows:

Syntax	Matches
iri	An IRI. A path of length one.
^elt	Inverse path (object to subject).
elt1 / elt2	A sequence path of elt1 followed by elt2.
elt1 | elt2	A alternative path of elt1 or elt2 (all possibilities are tried).
elt*	A path that connects the subject and object of the path by zero or more matches of elt.
elt+	A path that connects the subject and object of the path by one or more matches of elt.
elt?	A path that connects the subject and object of the path by zero or one matches of elt.
!iri or !(iri_1| … |iri_n)	Negated property set. An IRI which is not one of $iri_1 \dots iri_n$. !iri is short for !(iri).
!^iri or !(^iri_1| …|^iri_n)	Negated property set where the excluded matches are based on reversed path. That is, not one of $iri_1 \dots iri_n$ as reverse paths. !^iri is short for !(^iri).
!(iri_1| …|iri_j|^iri_{j+1}|… |^iri_n)|	A combination of forward and reverse properties in a negated property set.
(elt)	A group path elt, brackets control precedence.

This module is used internally by the SPARQL engine, but the property paths can also be used to query RDFLib Graphs directly.

Where possible the SPARQL syntax is mapped to Python operators, and property path objects can be constructed from existing URIRefs.

```
>>> from rdflib import Graph, Namespace
>>> from rdflib.namespace import FOAF
```

```
>>> ~FOAF.knows
Path(~http://xmlns.com/foaf/0.1/knows)
```

```
>>> FOAF.knows/FOAF.name
Path(http://xmlns.com/foaf/0.1/knows / http://xmlns.com/foaf/0.1/name)
```

```
>>> FOAF.name|FOAF.givenName
Path(http://xmlns.com/foaf/0.1/name | http://xmlns.com/foaf/0.1/givenName)
```

Modifiers (?, *, +) are done using * (the multiplication operator) and the strings '*', '?', '+', also defined as constants in this file.

```
>>> FOAF.knows*OneOrMore
Path(http://xmlns.com/foaf/0.1/knows+)
```

The path objects can also be used with the normal graph methods.

First some example data:

```
>>> g=Graph()
```

```
>>> g=g.parse(data='''
... @prefix : <ex:> .
...
... :a :p1 :c ; :p2 :f .
... :c :p2 :e ; :p3 :g .
... :g :p3 :h ; :p2 :j .
... :h :p3 :a ; :p2 :g .
...
... :q :px :q .
...
... ''', format='n3')
```

```
>>> e = Namespace('ex:')
```

Graph contains:

```
>>> (e.a, e.p1/e.p2, e.e) in g
True
```

Graph generator functions, triples, subjects, objects, etc. :

```
>>> list(g.objects(e.c, (e.p3*OneOrMore)/e.p2))
[rdflib.term.URIRef('ex:j'), rdflib.term.URIRef('ex:g'),
    rdflib.term.URIRef('ex:f')]
```

A more complete set of tests:

```
>>> list(eval_path(g, (None, e.p1/e.p2, None)))==[(e.a, e.e)]
True
>>> list(eval_path(g, (e.a, e.p1|e.p2, None)))==[(e.a,e.c), (e.a,e.f)]
True
>>> list(eval_path(g, (e.c, ~e.p1, None))) == [ (e.c, e.a) ]
True
>>> list(eval_path(g, (e.a, e.p1*ZeroOrOne, None))) == [(e.a, e.a), (e.a, e.c)]
True
>>> list(eval_path(g, (e.c, e.p3*OneOrMore, None))) == [
...     (e.c, e.g), (e.c, e.h), (e.c, e.a)]
True
>>> list(eval_path(g, (e.c, e.p3*ZeroOrMore, None))) == [(e.c, e.c),
...     (e.c, e.g), (e.c, e.h), (e.c, e.a)]
True
>>> list(eval_path(g, (e.a, -e.p1, None))) == [(e.a, e.f)]
True
>>> list(eval_path(g, (e.a, -(e.p1|e.p2), None))) == []
True
>>> list(eval_path(g, (e.g, -~e.p2, None))) == [(e.g, e.j)]
True
>>> list(eval_path(g, (e.e, ~(e.p1/e.p2), None))) == [(e.e, e.a)]
True
>>> list(eval_path(g, (e.a, e.p1/e.p3/e.p3, None))) == [(e.a, e.h)]
True
```

```
>>> list(eval_path(g, (e.q, e.px*OneOrMore, None)))
[(rdflib.term.URIRef('ex:q'), rdflib.term.URIRef('ex:q'))]
```

```
>>> list(eval_path(g, (None, e.p1|e.p2, e.c)))
[(rdflib.term.URIRef('ex:a'), rdflib.term.URIRef('ex:c'))]
```

```
>>> list(eval_path(g, (None, ~e.p1, e.a))) == [ (e.c, e.a) ]
True
>>> list(eval_path(g, (None, e.p1*ZeroOrOne, e.c)))
[(rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:c')),
 (rdflib.term.URIRef('ex:a'), rdflib.term.URIRef('ex:c'))]
```

```
>>> list(eval_path(g, (None, e.p3*OneOrMore, e.a)))
[(rdflib.term.URIRef('ex:h'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:g'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:a'))]
```

```
>>> list(eval_path(g, (None, e.p3*ZeroOrMore, e.a)))
[(rdflib.term.URIRef('ex:a'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:h'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:g'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:a'))]
```

```
>>> list(eval_path(g, (None, -e.p1, e.f))) == [(e.a, e.f)]
True
>>> list(eval_path(g, (None, -(e.p1|e.p2), e.c))) == []
True
>>> list(eval_path(g, (None, -~e.p2, e.j))) == [(e.g, e.j)]
True
>>> list(eval_path(g, (None, ~(e.p1/e.p2), e.a))) == [(e.e, e.a)]
True
>>> list(eval_path(g, (None, e.p1/e.p3/e.p3, e.h))) == [(e.a, e.h)]
True
```

```
>>> list(eval_path(g, (e.q, e.px*OneOrMore, None)))
[(rdflib.term.URIRef('ex:q'), rdflib.term.URIRef('ex:q'))]
```

```
>>> list(eval_path(g, (e.c, (e.p2|e.p3)*ZeroOrMore, e.j)))
[(rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:j'))]
```

No vars specified:

```
>>> sorted(list(eval_path(g, (None, e.p3*OneOrMore, None))))
[(rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:g')),
 (rdflib.term.URIRef('ex:c'), rdflib.term.URIRef('ex:h')),
 (rdflib.term.URIRef('ex:g'), rdflib.term.URIRef('ex:a')),
 (rdflib.term.URIRef('ex:g'), rdflib.term.URIRef('ex:h')),
 (rdflib.term.URIRef('ex:h'), rdflib.term.URIRef('ex:a'))]
```

class rdflib.paths.**AlternativePath**(**args*)

Bases: *Path*

Parameters
args (Union[*Path*, *URIRef*]) –

__init__(**args*)

Parameters
args (Union[*Path*, *URIRef*]) –

__module__ = 'rdflib.paths'

__repr__()
Return repr(self).

Return type
str

eval(*graph*, *subj=None*, *obj=None*)

Parameters

- **graph** (*Graph*) –
- **subj** (Optional[*Node*]) –
- **obj** (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

class rdflib.paths.**InvPath**(*arg*)
Bases: *Path*

Parameters
arg (Union[*Path*, *URIRef*]) –

__init__(*arg*)

Parameters
arg (Union[*Path*, *URIRef*]) –

__module__ = 'rdflib.paths'

__repr__()
Return repr(self).

Return type
str

eval(*graph*, *subj=None*, *obj=None*)

Parameters

- **graph** (*Graph*) –
- **subj** (Optional[*Node*]) –
- **obj** (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

class rdflib.paths.**MulPath**(*path*, *mod*)

Bases: *Path*

Parameters

- **path** (Union[*Path*, *URIRef*]) –
- **mod** (Literal['*', '+', '?']) –

__init__(*path*, *mod*)

Parameters

- **path** (Union[*Path*, *URIRef*]) –
- **mod** (Literal['*', '+', '?']) –

__module__ = 'rdflib.paths'

__repr__()

Return repr(self).

Return type
str

eval(*graph*, *subj=None*, *obj=None*, *first=True*)

Parameters

- **graph** (*Graph*) –
- **subj** (Optional[*Node*]) –
- **obj** (Optional[*Node*]) –
- **first** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

class rdflib.paths.**NegatedPath**(*arg*)

Bases: *Path*

Parameters
arg (Union[*AlternativePath*, *InvPath*, *URIRef*]) –

__init__(*arg*)

Parameters
arg (Union[*AlternativePath*, *InvPath*, *URIRef*]) –

__module__ = 'rdflib.paths'

__repr__()
Return repr(self).

Return type
str

eval(*graph*, *subj=None*, *obj=None*)

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

class rdflib.paths.Path
Bases: object

__annotations__ = {'__invert__': "Callable[['Path'], 'InvPath']", '__mul__': "Callable[['Path', str], 'MulPath']", '__neg__': "Callable[['Path'], 'NegatedPath']", '__or__': "Callable[['Path', Union['URIRef', 'Path']], 'AlternativePath']", '__truediv__': "Callable[['Path', Union['URIRef', 'Path']], 'SequencePath']"}

__dict__ = mappingproxy({'__module__': 'rdflib.paths', '__annotations__': {'__or__': "Callable[['Path', Union['URIRef', 'Path']], 'AlternativePath']", '__invert__': "Callable[['Path'], 'InvPath']", '__neg__': "Callable[['Path'], 'NegatedPath']", '__truediv__': "Callable[['Path', Union['URIRef', 'Path']], 'SequencePath']", '__mul__': "Callable[['Path', str], 'MulPath']"}, 'eval': <function Path.eval>, '__lt__': <function Path.__lt__>, '__dict__': <attribute '__dict__' of 'Path' objects>, '__weakref__': <attribute '__weakref__' of 'Path' objects>, '__doc__': None, '__gt__': <function _gt_from_lt>, '__le__': <function _le_from_lt>, '__ge__': <function _ge_from_lt>, '__invert__': <function inv_path>, '__neg__': <function neg_path>, '__mul__': <function mul_path>, '__or__': <function path_alternative>, '__truediv__': <function path_sequence>})

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

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

__invert__()
inverse path

Parameters
p (Union[*URIRef*, *Path*]) –

Return type
InvPath

__le__(*other*, *NotImplemented=NotImplemented*)

Return a <= b. Computed by @total_ordering from (a < b) or (a == b).

__lt__(*other*)

Return self<value.

Parameters
other (Any) –

Return type
bool

__module__ = 'rdflib.paths'

__mul__(*mul*)

cardinality path

Parameters

- **p** (Union[*URIRef*, *Path*]) –
- **mul** (Literal['*', '+', '?']) –

Return type
MulPath

__neg__()

negated path

Parameters
p (Union[*URIRef*, *AlternativePath*, *InvPath*]) –

Return type
NegatedPath

__or__(*other*)

alternative path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

__truediv__(*other*)

sequence path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

__weakref__

list of weak references to the object (if defined)

eval(*graph*, *subj=None*, *obj=None*)

Parameters

- **graph** (*Graph*) –
- **subj** (Optional[*Node*]) –
- **obj** (Optional[*Node*]) –

Return type
Iterator[Tuple[*Node*, *Node*]]

class rdflib.paths.**PathList**(*iterable=()*, */*)

Bases: list

__dict__ = mappingproxy({'__module__': 'rdflib.paths', '__dict__': <attribute '__dict__' of 'PathList' objects>, '__weakref__': <attribute '__weakref__' of 'PathList' objects>, '__doc__': None, '__annotations__': {}})

__module__ = 'rdflib.paths'

__weakref__
list of weak references to the object (if defined)

class rdflib.paths.**SequencePath**(**args*)

Bases: *Path*

Parameters
args (Union[*Path*, *URIRef*]) –

__init__(**args*)

Parameters
args (Union[*Path*, *URIRef*]) –

__module__ = 'rdflib.paths'

__repr__()
Return repr(self).

Return type
str

eval(*graph*, *subj=None*, *obj=None*)

Parameters
- **graph** (*Graph*) –
- **subj** (Optional[*Node*]) –
- **obj** (Optional[*Node*]) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

rdflib.paths.**evalPath**(*graph*, *t*)

Parameters
- **graph** (*Graph*) –
- **t** (Tuple[Optional[*Node*], Union[None, *Path*, *Node*], Optional[*Node*]]) –

Return type
Iterator[Tuple[*Node*, *Node*]]

rdflib.paths.**eval_path**(*graph*, *t*)

Parameters

- **graph** (*Graph*) –
- **t** (Tuple[Optional[*Node*], Union[None, *Path*, *Node*], Optional[*Node*]]) –

Return type
Iterator[Tuple[*Node*, *Node*]]

rdflib.paths.**inv_path**(*p*)

inverse path

Parameters
p (Union[*URIRef*, *Path*]) –

Return type
InvPath

rdflib.paths.**mul_path**(*p*, *mul*)

cardinality path

Parameters

- **p** (Union[*URIRef*, *Path*]) –
- **mul** (Literal['*', '+', '?']) –

Return type
MulPath

rdflib.paths.**neg_path**(*p*)

negated path

Parameters
p (Union[*URIRef*, *AlternativePath*, *InvPath*]) –

Return type
NegatedPath

rdflib.paths.**path_alternative**(*self*, *other*)

alternative path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

rdflib.paths.**path_sequence**(*self*, *other*)

sequence path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

rdflib.plugin module

Plugin support for rdf.

There are a number of plugin points for rdf: parser, serializer, store, query processor, and query result. Plugins can be registered either through setuptools entry_points or by calling rdf.plugin.register directly.

If you have a package that uses a setuptools based setup.py you can add the following to your setup:

```
entry_points = {
    'rdf.plugins.parser': [
        'nt =     rdf.plugins.parsers.ntriples:NTParser',
        ],
    'rdf.plugins.serializer': [
        'nt =     rdf.plugins.serializers.NTSerializer:NTSerializer',
        ],
    }
```

See the setuptools dynamic discovery of services and plugins for more information.

class rdflib.plugin.**PKGPlugin**(*name*, *kind*, *ep*)

Bases: *Plugin*[*PluginT*]

Parameters

- **name** (str) –
- **kind** (Type[TypeVar(PluginT)]) –
- **ep** (EntryPoint) –

__init__(*name*, *kind*, *ep*)

Parameters

- **name** (str) –
- **kind** (Type[TypeVar(PluginT)]) –
- **ep** (EntryPoint) –

__module__ = 'rdflib.plugin'

__orig_bases__ = (rdflib.plugin.Plugin[~PluginT],)

__parameters__ = (~PluginT,)

getClass()

Return type
Type[TypeVar(PluginT)]

class rdflib.plugin.**Plugin**(*name*, *kind*, *module_path*, *class_name*)

Bases: Generic[*PluginT*]

Parameters

- **name** (str) –
- **kind** (Type[TypeVar(PluginT)]) –
- **module_path** (str) –
- **class_name** (str) –

__dict__ = mappingproxy({'__module__': 'rdflib.plugin', '__init__': <function Plugin.__init__>, 'getClass': <function Plugin.getClass>, '__orig_bases__': (typing.Generic[~PluginT],), '__dict__': <attribute '__dict__' of 'Plugin' objects>, '__weakref__': <attribute '__weakref__' of 'Plugin' objects>, '__doc__': None, '__parameters__': (~PluginT,), '__annotations__': {'_class': 'Optional[Type[PluginT]]'}})

__init__(*name*, *kind*, *module_path*, *class_name*)

Parameters

- **name** (str) –
- **kind** (Type[TypeVar(PluginT)]) –
- **module_path** (str) –
- **class_name** (str) –

__module__ = 'rdflib.plugin'

__orig_bases__ = (typing.Generic[~PluginT],)

__parameters__ = (~PluginT,)

__weakref__

list of weak references to the object (if defined)

getClass()

Return type

Type[TypeVar(PluginT)]

exception rdflib.plugin.**PluginException**(*msg=None*)

Bases: *Error*

Parameters

msg (Optional[str]) –

__module__ = 'rdflib.plugin'

rdflib.plugin.**PluginT**

A generic type variable for plugins

alias of TypeVar('PluginT')

rdflib.plugin.**get**(*name*, *kind*)

Return the class for the specified (name, kind). Raises a PluginException if unable to do so.

Parameters

- **name** (str) –
- **kind** (Type[TypeVar(PluginT)]) –

Return type

Type[TypeVar(PluginT)]

rdflib.plugin.**plugins**(*name: Optional[str] = None*, *kind: Type[PluginT] = None*) → Iterator[*Plugin*[PluginT]]

rdflib.plugin.**plugins**(*name: Optional[str] = None*, *kind: None = None*) → Iterator[*Plugin*]

A generator of the plugins.

Pass in name and kind to filter... else leave None to match all.

Parameters

- **name** (Optional[str]) –
- **kind** (Optional[Type[TypeVar(PluginT)]]) –

Return type
Iterator[*Plugin*[TypeVar(PluginT)]]

rdflib.plugin.**register**(*name*, *kind*, *module_path*, *class_name*)

Register the plugin for (name, kind). The module_path and class_name should be the path to a plugin class.

Parameters

- **name** (str) –
- **kind** (Type[Any]) –

rdflib.query module

class rdflib.query.**EncodeOnlyUnicode**(*stream*)

Bases: object

This is a crappy work-around for http://bugs.python.org/issue11649

Parameters
stream (BinaryIO) –

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__doc__': '\n This is a crappy work-around for\n http://bugs.python.org/issue11649\n\n\n ', '__init__': <function EncodeOnlyUnicode.__init__>, 'write': <function EncodeOnlyUnicode.write>, '__getattr__': <function EncodeOnlyUnicode.__getattr__>, '__dict__': <attribute '__dict__' of 'EncodeOnlyUnicode' objects>, '__weakref__': <attribute '__weakref__' of 'EncodeOnlyUnicode' objects>, '__annotations__': {}})

__getattr__(*name*)

Parameters
name (str) –

Return type
Any

__init__(*stream*)

Parameters
stream (BinaryIO) –

__module__ = 'rdflib.query'

__weakref__

list of weak references to the object (if defined)

write(*arg*)

class rdflib.query.**Processor**(*graph*)

Bases: object

Query plugin interface.

This module is useful for those wanting to write a query processor that can plugin to rdf. If you are wanting to execute a query you likely want to do so through the Graph class query method.

Parameters
graph (*Graph*) –

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__doc__': '\n Query plugin interface.\n\n This module is useful for those wanting to write a query processor\n that can plugin to rdf. If you are wanting to execute a query you\n likely want to do so through the Graph class query method.\n\n ', '__init__': <function Processor.__init__>, 'query': <function Processor.query>, '__dict__': <attribute '__dict__' of 'Processor' objects>, '__weakref__': <attribute '__weakref__' of 'Processor' objects>, '__annotations__': {}})

__init__(*graph*)

Parameters
graph (*Graph*) –

__module__ = 'rdflib.query'

__weakref__

list of weak references to the object (if defined)

query(*strOrQuery*, *initBindings={}*, *initNs={}*, *DEBUG=False*)

Parameters

- **strOrQuery** (Union[str, *Query*]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **initNs** (Mapping[str, Any]) –
- **DEBUG** (bool) –

Return type
Mapping[str, Any]

class rdflib.query.**Result**(*type_*)

Bases: object

A common class for representing query result.

There is a bit of magic here that makes this appear like different Python objects, depending on the type of result.

If the type is “SELECT”, iterating will yield lists of ResultRow objects

If the type is “ASK”, iterating will yield a single bool (or bool(result) will return the same bool)

If the type is “CONSTRUCT” or “DESCRIBE” iterating will yield the triples.

len(result) also works.

Parameters
type_ (str) –

__bool__()

Return type
bool

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__doc__': '\n A common class for representing query result.\n\n There is a bit of magic here that makes this appear like different\n Python objects, depending on the type of result.\n\n If the type is "SELECT", iterating will yield lists of ResultRow objects\n\n If the type is "ASK", iterating will yield a single bool (or\n bool(result) will return the same bool)\n\n If the type is "CONSTRUCT" or "DESCRIBE" iterating will yield the\n triples.\n\n len(result) also works.\n\n ', '__init__': <function Result.__init__>, 'bindings': <property object>, 'parse': <staticmethod object>, 'serialize': <function Result.serialize>, '__len__': <function Result.__len__>, '__bool__': <function Result.__bool__>, '__iter__': <function Result.__iter__>, '__getattr__': <function Result.__getattr__>, '__eq__': <function Result.__eq__>, '__dict__': <attribute '__dict__' of 'Result' objects>, '__weakref__': <attribute '__weakref__' of 'Result' objects>, '__hash__': None, '__annotations__': {'vars': "Optional[List['Variable']]", '_bindings': "MutableSequence[Mapping['Variable', 'Identifier']]", '_genbindings': "Optional[Iterator[Mapping['Variable', 'Identifier']]]", 'askAnswer': 'Optional[bool]', 'graph': "Optional['Graph']"}})

__eq__(*other*)

Return self==value.

Parameters
other (Any) –

Return type
bool

__getattr__(*name*)

Parameters
name (str) –

Return type
Any

__hash__ = None

__init__(*type_*)

Parameters
type_ (str) –

__iter__()

Return type
Iterator[Union[Tuple[*Node*, *Node*, *Node*], bool, *ResultRow*]]

__len__()

Return type
int

__module__ = 'rdflib.query'

__weakref__

list of weak references to the object (if defined)

property bindings: MutableSequence[Mapping[*Variable*, *Identifier*]]

a list of variable bindings as dicts

Return type
MutableSequence[Mapping[*Variable*, *Identifier*]]

static parse(*source=None*, *format=None*, *content_type=None*, ***kwargs*)

Parameters

- **source** (Optional[IO]) –
- **format** (Optional[str]) –
- **content_type** (Optional[str]) –
- **kwargs** (Any) –

Return type
Result

serialize(*destination=None*, *encoding='utf-8'*, *format='xml'*, ***args*)

Serialize the query result.

The `format` argument determines the Serializer class to use.

- csv: *CSVResultSerializer*
- json: *JSONResultSerializer*
- txt: *TXTResultSerializer*
- xml: *XMLResultSerializer*

Parameters

- **destination** (Union[str, IO, None]) – Path of file output or BufferedIOBase object to write the output to.
- **encoding** (str) – Encoding of output.
- **format** (str) – One of [‘csv’, ‘json’, ‘txt’, xml’]
- **args** (Any) –

Return type
Optional[bytes]

Returns
bytes

vars: Optional[List[*Variable*]]

variables contained in the result.

exception rdflib.query.**ResultException**

Bases: Exception

__module__ = 'rdflib.query'

__weakref__

list of weak references to the object (if defined)

class rdflib.query.**ResultParser**

Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__init__': <function ResultParser.__init__>, 'parse': <function ResultParser.parse>, '__dict__': <attribute '__dict__' of 'ResultParser' objects>, '__weakref__': <attribute '__weakref__' of 'ResultParser' objects>, '__doc__': None, '__annotations__': {}})

__init__()

__module__ = 'rdflib.query'

__weakref__

list of weak references to the object (if defined)

parse(*source*, ***kwargs*)

return a Result object

Parameters

- **source** (IO) –
- **kwargs** (Any) –

Return type

Result

class rdflib.query.**ResultRow**(*values: Mapping[*Variable, Identifier*]*, *labels: List[*Variable*]*)

Bases: Tuple[Identifier, ...]

a single result row allows accessing bindings as attributes or with []

```
>>> from rdflib import URIRef, Variable
>>> rr=ResultRow({ Variable('a'): URIRef('urn:cake') }, [Variable('a')])
```

```
>>> rr[0]
rdflib.term.URIRef(u'urn:cake')
>>> rr[1]
Traceback (most recent call last):
    ...
IndexError: tuple index out of range
```

```
>>> rr.a
rdflib.term.URIRef(u'urn:cake')
>>> rr.b
Traceback (most recent call last):
    ...
AttributeError: b
```

```
>>> rr['a']
rdflib.term.URIRef(u'urn:cake')
>>> rr['b']
Traceback (most recent call last):
    ...
KeyError: 'b'
```

```
>>> rr[Variable('a')]
rdflib.term.URIRef(u'urn:cake')
```

New in version 4.0.

__annotations__ = {'labels': 'Mapping[str, int]'}

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__annotations__': {'labels': 'Mapping[str, int]'}, '__doc__': "\n a single result row\n allows accessing bindings as attributes or with []\n\n >>> from rdflib import URIRef, Variable\n >>> rr=ResultRow({ Variable('a'): URIRef('urn:cake') }, [Variable('a')])\n\n >>> rr[0]\n rdflib.term.URIRef(u'urn:cake')\n >>> rr[1]\n Traceback (most recent call last):\n ...\n IndexError: tuple index out of range\n\n >>> rr.a\n rdflib.term.URIRef(u'urn:cake')\n >>> rr.b\n Traceback (most recent call last):\n ...\n AttributeError: b\n\n >>> rr['a']\n rdflib.term.URIRef(u'urn:cake')\n >>> rr['b']\n Traceback (most recent call last):\n ...\n KeyError: 'b'\n\n >>> rr[Variable('a')]\n rdflib.term.URIRef(u'urn:cake')\n\n .. versionadded:: 4.0\n\n ", '__new__': <staticmethod object>, '__getattr__': <function ResultRow.__getattr__>, '__getitem__': <function ResultRow.__getitem__>, 'get': <function ResultRow.get>, 'asdict': <function ResultRow.asdict>, '__orig_bases__': (typing.Tuple[ForwardRef('Identifier'), ...],), '__dict__': <attribute '__dict__' of 'ResultRow' objects>, '__parameters__': ()})

__getattr__(*name*)

Parameters
name (str) –

Return type
Identifier

__getitem__(*name*)

Return self[key].

Parameters
name (Union[str, int, Any]) –

Return type
Identifier

__module__ = 'rdflib.query'

static __new__(*cls*, *values*, *labels*)

Parameters

- **values** (Mapping[*Variable*, *Identifier*]) –
- **labels** (List[*Variable*]) –

__orig_bases__ = (typing.Tuple[ForwardRef('Identifier'), ...],)

__parameters__ = ()

asdict()

Return type
Dict[str, *Identifier*]

get(*name: str*, *default:* Identifier) → *Identifier*

get(*name: str*, *default: Optional[*Identifier*] = None*) → Optional[*Identifier*]

Parameters

- **name** (str) –

- **default** (Optional[*Identifier*]) –

Return type
Optional[*Identifier*]

labels: Mapping[str, int]

class rdflib.query.**ResultSerializer**(*result*)

Bases: object

Parameters
result (*Result*) –

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__init__': <function ResultSerializer.__init__>, 'serialize': <function ResultSerializer.serialize>, '__dict__': <attribute '__dict__' of 'ResultSerializer' objects>, '__weakref__': <attribute '__weakref__' of 'ResultSerializer' objects>, '__doc__': None, '__annotations__': {}})

__init__(*result*)

Parameters
result (*Result*) –

__module__ = 'rdflib.query'

__weakref__
list of weak references to the object (if defined)

serialize(*stream*, *encoding='utf-8'*, ***kwargs*)
return a string properly serialized

Parameters

- **stream** (IO) –
- **encoding** (str) –
- **kwargs** (Any) –

Return type
None

class rdflib.query.**UpdateProcessor**(*graph*)

Bases: object

Update plugin interface.

This module is useful for those wanting to write an update processor that can plugin to rdflib. If you are wanting to execute an update statement you likely want to do so through the Graph class update method.

New in version 4.0.

Parameters
graph (*Graph*) –

__dict__ = mappingproxy({'__module__': 'rdflib.query', '__doc__': '\n Update plugin interface.\n\n This module is useful for those wanting to write an update\n processor that can plugin to rdflib. If you are wanting to execute\n an update statement you likely want to do so through the Graph\n class update method.\n\n .. versionadded:: 4.0\n\n ', '__init__': <function UpdateProcessor.__init__>, 'update': <function UpdateProcessor.update>, '__dict__': <attribute '__dict__' of 'UpdateProcessor' objects>, '__weakref__': <attribute '__weakref__' of 'UpdateProcessor' objects>, '__annotations__': {}})

__init__(*graph*)

Parameters
graph (*Graph*) –

__module__ = 'rdflib.query'

__weakref__
list of weak references to the object (if defined)

update(*strOrQuery*, *initBindings={}*, *initNs={}*)

Parameters

- **strOrQuery** (Union[str, *Update*]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **initNs** (Mapping[str, Any]) –

Return type
None

rdflib.resource module

The *Resource* class wraps a *Graph* and a resource reference (i.e. a *rdflib.term.URIRef* or *rdflib.term.BNode*) to support a resource-oriented way of working with a graph.

It contains methods directly corresponding to those methods of the Graph interface that relate to reading and writing data. The difference is that a Resource also binds a resource identifier, making it possible to work without tracking both the graph and a current subject. This makes for a "resource oriented" style, as compared to the triple orientation of the Graph API.

Resulting generators are also wrapped so that any resource reference values (*rdflib.term.URIRef* and *rdflib.term.BNode*) are in turn wrapped as Resources. (Note that this behaviour differs from the corresponding methods in *Graph*, where no such conversion takes place.)

Basic Usage Scenario

Start by importing things we need and define some namespaces:

```
>>> from rdflib import *
>>> FOAF = Namespace("http://xmlns.com/foaf/0.1/")
>>> CV = Namespace("http://purl.org/captsolo/resume-rdf/0.2/cv#")
```

Load some RDF data:

```
>>> graph = Graph().parse(format='n3', data='''
... @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
... @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
... @prefix foaf: <http://xmlns.com/foaf/0.1/> .
... @prefix cv: <http://purl.org/captsolo/resume-rdf/0.2/cv#> .
...
... @base <http://example.org/> .
...
```

(continues on next page)

(continued from previous page)

```
... </person/some1#self> a foaf:Person;
...     rdfs:comment "Just a Python & RDF hacker."@en;
...     foaf:depiction </images/person/some1.jpg>;
...     foaf:homepage <http://example.net/>;
...     foaf:name "Some Body" .
...
... </images/person/some1.jpg> a foaf:Image;
...     rdfs:label "some 1"@en;
...     rdfs:comment "Just an image"@en;
...     foaf:thumbnail </images/person/some1-thumb.jpg> .
...
... </images/person/some1-thumb.jpg> a foaf:Image .
...
... [] a cv:CV;
...     cv:aboutPerson </person/some1#self>;
...     cv:hasWorkHistory [ cv:employedIn </#company>;
...             cv:startDate "2009-09-04"^^xsd:date ] .
... ''')
```

Create a Resource:

```
>>> person = Resource(
...     graph, URIRef("http://example.org/person/some1#self"))
```

Retrieve some basic facts:

```
>>> person.identifier
rdflib.term.URIRef(u'http://example.org/person/some1#self')

>>> person.value(FOAF.name)
rdflib.term.Literal(u'Some Body')

>>> person.value(RDFS.comment)
rdflib.term.Literal(u'Just a Python & RDF hacker.', lang=u'en')
```

Resources can be sliced (like graphs, but the subject is fixed):

```
>>> for name in person[FOAF.name]:
...     print(name)
Some Body
>>> person[FOAF.name : Literal("Some Body")]
True
```

Resources as unicode are represented by their identifiers as unicode:

```
>>> %(unicode)s(person)
u'Resource(http://example.org/person/some1#self'
```

Resource references are also Resources, so you can easily get e.g. a qname for the type of a resource, like:

```
>>> person.value(RDF.type).qname()
u'foaf:Person'
```

Or for the predicates of a resource:

```
>>> sorted(
...     p.qname() for p in person.predicates()
... )
[u'foaf:depiction', u'foaf:homepage',
 u'foaf:name', u'rdf:type', u'rdfs:comment']
```

Follow relations and get more data from their Resources as well:

```
>>> for pic in person.objects(FOAF.depiction):
...     print(pic.identifier)
...     print(pic.value(RDF.type).qname())
...     print(pic.value(FOAF.thumbnail).identifier)
http://example.org/images/person/some1.jpg
foaf:Image
http://example.org/images/person/some1-thumb.jpg

>>> for cv in person.subjects(CV.aboutPerson):
...     work = list(cv.objects(CV.hasWorkHistory))[0]
...     print(work.value(CV.employedIn).identifier)
...     print(work.value(CV.startDate))
http://example.org/#company
2009-09-04
```

It's just as easy to work with the predicates of a resource:

```
>>> for s, p in person.subject_predicates():
...     print(s.value(RDF.type).qname())
...     print(p.qname())
...     for s, o in p.subject_objects():
...         print(s.value(RDF.type).qname())
...         print(o.value(RDF.type).qname())
cv:CV
cv:aboutPerson
cv:CV
foaf:Person
```

This is useful for e.g. inspection:

```
>>> thumb_ref = URIRef("http://example.org/images/person/some1-thumb.jpg")
>>> thumb = Resource(graph, thumb_ref)
>>> for p, o in thumb.predicate_objects():
...     print(p.qname())
...     print(o.qname())
rdf:type
foaf:Image
```

Schema Example

With this artificial schema data:

```
>>> graph = Graph().parse(format='n3', data='''
... @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
... @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
... @prefix owl: <http://www.w3.org/2002/07/owl#> .
... @prefix v: <http://example.org/def/v#> .
...
... v:Artifact a owl:Class .
...
... v:Document a owl:Class;
...     rdfs:subClassOf v:Artifact .
...
... v:Paper a owl:Class;
...     rdfs:subClassOf v:Document .
...
... v:Choice owl:oneOf (v:One v:Other) .
...
... v:Stuff a rdf:Seq; rdf:_1 v:One; rdf:_2 v:Other .
...
... ''')
```

From this class:

```
>>> artifact = Resource(graph, URIRef("http://example.org/def/v#Artifact"))
```

we can get at subclasses:

```
>>> subclasses = list(artifact.transitive_subjects(RDFS.subClassOf))
>>> [c.qname() for c in subclasses]
[u'v:Artifact', u'v:Document', u'v:Paper']
```

and superclasses from the last subclass:

```
>>> [c.qname() for c in subclasses[-1].transitive_objects(RDFS.subClassOf)]
[u'v:Paper', u'v:Document', u'v:Artifact']
```

Get items from the Choice:

```
>>> choice = Resource(graph, URIRef("http://example.org/def/v#Choice"))
>>> [it.qname() for it in choice.value(OWL.oneOf).items()]
[u'v:One', u'v:Other']
```

On add, other resources are auto-unboxed:

```
>>> paper = Resource(graph, URIRef("http://example.org/def/v#Paper"))
>>> paper.add(RDFS.subClassOf, artifact)
>>> artifact in paper.objects(RDFS.subClassOf) # checks Resource instance
True
>>> (paper._identifier, RDFS.subClassOf, artifact._identifier) in graph
True
```

Technical Details

Comparison is based on graph and identifier:

```
>>> g1 = Graph()
>>> t1 = Resource(g1, URIRef("http://example.org/thing"))
>>> t2 = Resource(g1, URIRef("http://example.org/thing"))
>>> t3 = Resource(g1, URIRef("http://example.org/other"))
>>> t4 = Resource(Graph(), URIRef("http://example.org/other"))

>>> t1 is t2
False

>>> t1 == t2
True
>>> t1 != t2
False

>>> t1 == t3
False
>>> t1 != t3
True

>>> t3 != t4
True

>>> t3 < t1 and t1 > t3
True
>>> t1 >= t1 and t1 >= t3
True
>>> t1 <= t1 and t3 <= t1
True

>>> t1 < t1 or t1 < t3 or t3 > t1 or t3 > t3
False
```

Hash is computed from graph and identifier:

```
>>> g1 = Graph()
>>> t1 = Resource(g1, URIRef("http://example.org/thing"))

>>> hash(t1) == hash(Resource(g1, URIRef("http://example.org/thing")))
True

>>> hash(t1) == hash(Resource(Graph(), t1.identifier))
False
>>> hash(t1) == hash(Resource(Graph(), URIRef("http://example.org/thing")))
False
```

The Resource class is suitable as a base class for mapper toolkits. For example, consider this utility for accessing RDF properties via qname-like attributes:

```
>>> class Item(Resource):
```

(continues on next page)

(continued from previous page)

```
...
...     def __getattr__(self, p):
...         return list(self.objects(self._to_ref(*p.split('_', 1))))
...
...     def _to_ref(self, pfx, name):
...         return URIRef(self._graph.store.namespace(pfx) + name)
```

It works as follows:

```
>>> graph = Graph().parse(format='n3', data='''
... @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
... @prefix foaf: <http://xmlns.com/foaf/0.1/> .
...
... @base <http://example.org/> .
... </person/some1#self>
...     foaf:name "Some Body";
...     foaf:depiction </images/person/some1.jpg> .
... </images/person/some1.jpg> rdfs:comment "Just an image"@en .
... ''')

>>> person = Item(graph, URIRef("http://example.org/person/some1#self"))

>>> print(person.foaf_name[0])
Some Body
```

The mechanism for wrapping references as resources cooperates with subclasses. Therefore, accessing referenced resources automatically creates new `Item` objects:

```
>>> isinstance(person.foaf_depiction[0], Item)
True

>>> print(person.foaf_depiction[0].rdfs_comment[0])
Just an image
```

class rdflib.resource.**Resource**(*graph*, *subject*)

Bases: `object`

__dict__ = mappingproxy({'__module__': 'rdflib.resource', '__init__': <function Resource.__init__>, 'graph': <property object>, 'identifier': <property object>, '__hash__': <function Resource.__hash__>, '__eq__': <function Resource.__eq__>, '__ne__': <function Resource.__ne__>, '__lt__': <function Resource.__lt__>, '__gt__': <function Resource.__gt__>, '__le__': <function Resource.__le__>, '__ge__': <function Resource.__ge__>, '__unicode__': <function Resource.__unicode__>, 'add': <function Resource.add>, 'remove': <function Resource.remove>, 'set': <function Resource.set>, 'subjects': <function Resource.subjects>, 'predicates': <function Resource.predicates>, 'objects': <function Resource.objects>, 'subject_predicates': <function Resource.subject_predicates>, 'subject_objects': <function Resource.subject_objects>, 'predicate_objects': <function Resource.predicate_objects>, 'value': <function Resource.value>, 'items': <function Resource.items>, 'transitive_objects': <function Resource.transitive_objects>, 'transitive_subjects': <function Resource.transitive_subjects>, 'qname': <function Resource.qname>, '_resource_pairs': <function Resource._resource_pairs>, '_resource_triples': <function Resource._resource_triples>, '_resources': <function Resource._resources>, '_cast': <function Resource._cast>, '__iter__': <function Resource.__iter__>, '__getitem__': <function Resource.__getitem__>, '__setitem__': <function Resource.__setitem__>, '_new': <function Resource._new>, '__str__': <function Resource.__str__>, '__repr__': <function Resource.__repr__>, '__dict__': <attribute '__dict__' of 'Resource' objects>, '__weakref__': <attribute '__weakref__' of 'Resource' objects>, '__doc__': None, '__annotations__': {}})

__eq__(*other*)

Return self==value.

__ge__(*other*)

Return self>=value.

__getitem__(*item*)

__gt__(*other*)

Return self>value.

__hash__()

Return hash(self).

__init__(*graph*, *subject*)

__iter__()

__le__(*other*)

Return self<=value.

__lt__(*other*)

Return self<value.

__module__ = 'rdflib.resource'

__ne__(*other*)

Return self!=value.

__repr__()

Return repr(self).

__setitem__(*item*, *value*)

__str__()

Return str(self).

__unicode__()

__weakref__

list of weak references to the object (if defined)

add(*p*, *o*)

property graph

property identifier

items()

objects(*predicate=None*)

predicate_objects()

predicates(*o=None*)

qname()

remove(*p*, *o=None*)

set(*p*, *o*)

subject_objects()

subject_predicates()

subjects(*predicate=None*)

transitive_objects(*predicate*, *remember=None*)

transitive_subjects(*predicate*, *remember=None*)

value(*p=rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *o=None*, *default=None*, *any=True*)

rdflib.serializer module

class rdflib.serializer.**Serializer**(*store*)

Bases: object

Parameters
store (*Graph*) –

__dict__ = mappingproxy({'__module__': 'rdflib.serializer', '__init__': <function Serializer.__init__>, 'serialize': <function Serializer.serialize>, 'relativize': <function Serializer.relativize>, '__dict__': <attribute '__dict__' of 'Serializer' objects>, '__weakref__': <attribute '__weakref__' of 'Serializer' objects>, '__doc__': None, '__annotations__': {'store': "'Graph'", 'encoding': 'str', 'base': 'Optional[str]'}})

__init__(*store*)

Parameters
store (*Graph*) –

__module__ = 'rdflib.serializer'

__weakref__
list of weak references to the object (if defined)

relativize(*uri*)

Parameters
uri (TypeVar(_StrT, bound= str)) –

Return type
Union[TypeVar(_StrT, bound= str), *URIRef*]

serialize(*stream*, *base=None*, *encoding=None*, ***args*)
Abstract method

Parameters
- **stream** (IO[bytes]) –
- **base** (Optional[str]) –
- **encoding** (Optional[str]) –
- **args** (Any) –

Return type
None

rdflib.store module

class rdflib.store.**NodePickler**
Bases: object

__dict__ = mappingproxy({'__module__': 'rdflib.store', '__init__': <function NodePickler.__init__>, '_get_ids': <function NodePickler._get_ids>, 'register': <function NodePickler.register>, 'loads': <function NodePickler.loads>, 'dumps': <function NodePickler.dumps>, '__getstate__': <function NodePickler.__getstate__>, '__setstate__': <function NodePickler.__setstate__>, '__dict__': <attribute '__dict__' of 'NodePickler' objects>, '__weakref__': <attribute '__weakref__' of 'NodePickler' objects>, '__doc__': None, '__annotations__': {'_objects': 'Dict[str, Any]', '_ids': 'Dict[Any, str]'}})

__getstate__()

Return type
Mapping[str, Any]

__init__()

__module__ = 'rdflib.store'

__setstate__(*state*)

Parameters
state (Mapping[str, Any]) –

Return type
None

__weakref__
list of weak references to the object (if defined)

dumps(*obj*, *protocol=None*, *bin=None*)

Parameters

- **obj** (*Node*) –
- **protocol** (Optional[Any]) –
- **bin** (Optional[Any]) –

loads(*s*)

Parameters
s (bytes) –

Return type
Node

register(*object*, *id*)

Parameters

- **object** (Any) –
- **id** (str) –

Return type
None

class rdflib.store.Store(*configuration=None*, *identifier=None*)
Bases: object

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__annotations__ = {'context_aware': 'bool', 'formula_aware': 'bool', 'graph_aware': 'bool', 'transaction_aware': 'bool'}

__dict__ = mappingproxy({'__module__': 'rdflib.store', '__annotations__': {'context_aware': 'bool', 'formula_aware': 'bool', 'transaction_aware': 'bool', 'graph_aware': 'bool', '__node_pickler': 'Optional[NodePickler]'}, 'context_aware': False, 'formula_aware': False, 'transaction_aware': False, 'graph_aware': False, '__init__': <function Store.__init__>, 'node_pickler': <property object>, 'create': <function Store.create>, 'open': <function Store.open>, 'close': <function Store.close>, 'destroy': <function Store.destroy>, 'gc': <function Store.gc>, 'add': <function Store.add>, 'addN': <function Store.addN>, 'remove': <function Store.remove>, 'triples_choices': <function Store.triples_choices>, 'triples': <function Store.triples>, '__len__': <function Store.__len__>, 'contexts': <function Store.contexts>, 'query': <function Store.query>, 'update': <function Store.update>, 'bind': <function Store.bind>, 'prefix': <function Store.prefix>, 'namespace': <function Store.namespace>, 'namespaces': <function Store.namespaces>, 'commit': <function Store.commit>, 'rollback': <function Store.rollback>, 'add_graph': <function Store.add_graph>, 'remove_graph': <function Store.remove_graph>, '__dict__': <attribute '__dict__' of 'Store' objects>, '__weakref__': <attribute '__weakref__' of 'Store' objects>, '__doc__': None})

__init__(*configuration=None*, *identifier=None*)

identifier: URIRef of the Store. Defaults to CWD configuration: string containing information open can use to connect to datastore.

Parameters

- **configuration** (Optional[str]) –
- **identifier** (Optional[*Identifier*]) –

__len__(*context=None*)

Number of statements in the store. This should only account for non- quoted (asserted) statements if the context is not specified, otherwise it should return the number of statements in the formula or context given.

Parameters

context (Optional[*Graph*]) – a graph instance to query or None

Return type

int

__module__ = 'rdflib.store'

__weakref__

list of weak references to the object (if defined)

add(*triple*, *context*, *quoted=False*)

Adds the given statement to a specific context or to the model. The quoted argument is interpreted by formula-aware stores to indicate this statement is quoted/hypothetical It should be an error to not specify a context and have the quoted argument be True. It should also be an error for the quoted argument to be True when the store is not formula-aware.

Parameters

- **triple** (Tuple[*Node*, *Node*, *Node*]) –
- **context** (*Graph*) –
- **quoted** (bool) –

Return type

None

addN(*quads*)

Adds each item in the list of statements to a specific context. The quoted argument is interpreted by formula-aware stores to indicate this statement is quoted/hypothetical. Note that the default implementation is a redirect to add

Parameters
quads (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
None

add_graph(*graph*)

Add a graph to the store, no effect if the graph already exists. :type graph: *Graph* :param graph: a Graph instance

Return type
None

bind(*prefix*, *namespace*, *override=True*)

Parameters

- **override** (bool) – rebind, even if the given namespace is already bound to another prefix.
- **prefix** (str) –
- **namespace** (*URIRef*) –

Return type
None

close(*commit_pending_transaction=False*)

This closes the database connection. The commit_pending_transaction parameter specifies whether to commit all pending transactions before closing (if the store is transactional).

Parameters
commit_pending_transaction (bool) –

Return type
None

commit()

Return type
None

context_aware: bool = False

contexts(*triple=None*)

Generator over all contexts in the graph. If triple is specified, a generator over all contexts the triple is in.

if store is graph_aware, may also return empty contexts

Return type
Generator[*Graph*, None, None]

Returns
a generator over Nodes

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

create(*configuration*)

Parameters
configuration (str) –

Return type
None

destroy(*configuration*)

This destroys the instance of the store identified by the configuration string.

Parameters
configuration (str) –

Return type
None

formula_aware: bool = False

gc()

Allows the store to perform any needed garbage collection

Return type
None

graph_aware: bool = False

namespace(*prefix*)

Parameters
prefix (str) –

Return type
Optional[*URIRef*]

namespaces()

Return type
Iterator[Tuple[str, *URIRef*]]

property node_pickler: *NodePickler*

Return type
NodePickler

open(*configuration*, *create=False*)

Opens the store specified by the configuration string. If create is True a store will be created if it does not already exist. If create is False and a store does not already exist an exception is raised. An exception is also raised if a store exists, but there is insufficient permissions to open the store. This should return one of: VALID_STORE, CORRUPTED_STORE, or NO_STORE

Parameters

- **configuration** (str) –
- **create** (bool) –

Return type
Optional[int]

prefix(*namespace*)

Parameters
namespace (*URIRef*) –

Return type
Optional[str]

query(*query*, *initNs*, *initBindings*, *queryGraph*, ***kwargs*)

If stores provide their own SPARQL implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **query** (Union[*Query*, str]) –
- **initNs** (Mapping[str, Any]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **queryGraph** (str) –
- **kwargs** (Any) –

Return type
Result

remove(*triple*, *context=None*)

Remove the set of triples matching the pattern from the store

Parameters

- **triple** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –
- **context** (Optional[*Graph*]) –

Return type
None

remove_graph(*graph*)

Remove a graph from the store, this should also remove all triples in the graph

Parameters

- **graphid** – a Graph instance
- **graph** (*Graph*) –

Return type
None

rollback()

Return type
None

transaction_aware: bool = False

triples(*triple_pattern*, *context=None*)

A generator over all the triples matching the pattern. Pattern can include any objects for used for comparing against nodes in the store, for example, REGEXTerm, URIRef, Literal, BNode, Variable, Graph, QuotedGraph, Date? DateRange?

Parameters

- **context** (Optional[*Graph*]) – A conjunctive query can be indicated by either providing a value of None, or a specific context can be queries by passing a Graph instance (if store is context aware).
- **triple_pattern** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type

Iterator[Tuple[Tuple[*Node*, *Node*, *Node*], Iterator[Optional[*Graph*]]]]

triples_choices(*triple*, *context=None*)

A variant of triples that can take a list of terms instead of a single term in any slot. Stores can implement this to optimize the response time from the default 'fallback' implementation, which will iterate over each term in the list and dispatch to triples

Parameters

- **triple** (Union[Tuple[List[*Node*], *Node*, *Node*], Tuple[*Node*, List[*Node*], *Node*], Tuple[*Node*, *Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type

Generator[Tuple[Tuple[*Node*, *Node*, *Node*], Iterator[Optional[*Graph*]]], None, None]

update(*update*, *initNs*, *initBindings*, *queryGraph*, ***kwargs*)

If stores provide their own (SPARQL) Update implementation, override this.

queryGraph is None, a URIRef or '__UNION__' If None the graph is specified in the query-string/object If URIRef it specifies the graph to query, If '__UNION__' the union of all named graphs should be queried (This is used by ConjunctiveGraphs Values other than None obviously only makes sense for context-aware stores.)

Parameters

- **update** (Union[*Update*, str]) –
- **initNs** (Mapping[str, Any]) –
- **initBindings** (Mapping[str, *Identifier*]) –
- **queryGraph** (str) –
- **kwargs** (Any) –

Return type

None

class rdflib.store.**StoreCreatedEvent**(***kw*)

Bases: *Event*

This event is fired when the Store is created, it has the following attribute:

- configuration: string used to create the store

__module__ = 'rdflib.store'

class rdflib.store.**TripleAddedEvent**(***kw*)

Bases: *Event*

This event is fired when a triple is added, it has the following attributes:

- the `triple` added to the graph
- the `context` of the triple, if any
- the `graph` to which the triple was added

__module__ = 'rdflib.store'

class rdflib.store.**TripleRemovedEvent**(***kw*)

Bases: *Event*

This event is fired when a triple is removed, it has the following attributes:

- the `triple` removed from the graph
- the `context` of the triple, if any
- the `graph` from which the triple was removed

__module__ = 'rdflib.store'

rdflib.term module

This module defines the different types of terms. Terms are the kinds of objects that can appear in a quoted/asserted triple. This includes those that are core to RDF:

- *Blank Nodes*
- *URI References*
- *Literals* (which consist of a literal value,datatype and language tag)

Those that extend the RDF model into N3:

- *Formulae*
- *Universal Quantifications (Variables)*

And those that are primarily for matching against 'Nodes' in the underlying Graph:

- REGEX Expressions
- Date Ranges
- Numerical Ranges

class rdflib.term.**BNode**(*value: ~typing.Optional[str] = None*, *_sn_gen: ~typing.Callable[[], str] = <function _serial_number_generator.<locals>._generator>*, *_prefix: str = 'N'*)

Bases: *IdentifiedNode*

RDF 1.1's Blank Nodes Section: https://www.w3.org/TR/rdf11-concepts/#section-blank-nodes

Blank Nodes are local identifiers for unnamed nodes in RDF graphs that are used in some concrete RDF syntaxes or RDF store implementations. They are always locally scoped to the file or RDF store, and are not persistent or portable identifiers for blank nodes. The identifiers for Blank Nodes are not part of the RDF abstract syntax, but are entirely dependent on particular concrete syntax or implementation (such as Turtle, JSON-LD).

—

RDFLib's BNode class makes unique IDs for all the Blank Nodes in a Graph but you should *never* expect, or reply on, BNodes' IDs to match across graphs, or even for multiple copies of the same graph, if they are regenerated from some non-RDFLib source, such as loading from RDF data.

__module__ = 'rdflib.term'

static __new__(*cls*, *value=None*, *_sn_gen=<function _serial_number_generator.<locals>._generator>*, *_prefix='N'*)

only store implementations should pass in a value

Parameters

- **value** (Optional[str]) –
- **_sn_gen** (Callable[[], str]) –
- **_prefix** (str) –

Return type
BNode

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*BNode*], Tuple[str]]

__repr__()

Return repr(self).

Return type
str

__slots__ = ()

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

skolemize(*authority=None*, *basepath=None*)

Create a URIRef "skolem" representation of the BNode, in accordance with http://www.w3.org/TR/rdf11-concepts/#section-skolemization

New in version 4.0.

Parameters

- **authority** (Optional[str]) –
- **basepath** (Optional[str]) –

Return type
URIRef

class rdflib.term.**IdentifiedNode**(*value: str*)

Bases: *Identifier*

An abstract class, primarily defined to identify Nodes that are not Literals.

The name "Identified Node" is not explicitly defined in the RDF specification, but can be drawn from this section: https://www.w3.org/TR/rdf-concepts/#section-URI-Vocabulary

__dict__ = mappingproxy({'__module__': 'rdflib.term', '__doc__': '\n An abstract class, primarily defined to identify Nodes that are not Literals.\n\n The name "Identified Node" is not explicitly defined in the RDF specification, but can be drawn from this section:\nhttps://www.w3.org/TR/rdf-concepts/#section-URI-Vocabulary\n ', '__getnewargs__': <function IdentifiedNode.__getnewargs__>, 'toPython': <function IdentifiedNode.toPython>, '__dict__': <attribute '__dict__' of 'IdentifiedNode' objects>, '__weakref__': <attribute '__weakref__' of 'IdentifiedNode' objects>, '__annotations__': {}})

__getnewargs__()

Return type
Tuple[str]

__module__ = 'rdflib.term'

__weakref__
list of weak references to the object (if defined)

toPython()

Return type
str

class rdflib.term.**Identifier**(*value: str*)

Bases: *Node*, str

See http://www.w3.org/2002/07/rdf-identifer-terminology/ regarding choice of terminology.

__eq__(*other*)
Equality for Nodes.

```
>>> BNode("foo")==None
False
>>> BNode("foo")==URIRef("foo")
False
>>> URIRef("foo")==BNode("foo")
False
>>> BNode("foo")!=URIRef("foo")
True
>>> URIRef("foo")!=BNode("foo")
True
>>> Variable('a')!=URIRef('a')
True
>>> Variable('a')!=Variable('a')
False
```

Parameters
other (Any) –

Return type
bool

__ge__(*other*)

Return self>=value.

Parameters
other (Any) –

Return type
bool

__gt__(*other*)

This implements ordering for Nodes,

This tries to implement this: http://www.w3.org/TR/sparql11-query/#modOrderBy

Variables are not included in the SPARQL list, but they are greater than BNodes and smaller than everything else

Parameters
other (Any) –

Return type
bool

__hash__()

Return hash(self).

__le__(*other*)

Return self<=value.

Parameters
other (Any) –

Return type
bool

__lt__(*other*)

Return self<value.

Parameters
other (Any) –

Return type
bool

__module__ = 'rdflib.term'

__ne__(*other*)

Return self!=value.

Parameters
other (Any) –

Return type
bool

static __new__(*cls*, *value*)

Parameters
value (str) –

Return type
Identifier

`__slots__ = ()`

eq(*other*)

A "semantic"/interpreted equality function, by default, same as __eq__

Parameters
other (Any) –

Return type
bool

neq(*other*)

A "semantic"/interpreted not equal function, by default, same as __ne__

Parameters
other (Any) –

Return type
bool

startswith(*prefix*, *start=Ellipsis*, *end=Ellipsis*)

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

Parameters
prefix (str) –

Return type
bool

class rdflib.term.**Literal**(*lexical_or_value: Any*, *lang: Optional[str] = None*, *datatype: Optional[str] = None*, *normalize: Optional[bool] = None*)

Bases: *Identifier*

RDF 1.1's Literals Section: http://www.w3.org/TR/rdf-concepts/#section-Graph-Literal

Literals are used for values such as strings, numbers, and dates.

A literal in an RDF graph consists of two or three elements:

- a lexical form, being a Unicode string, which SHOULD be in Normal Form C
- a datatype IRI, being an IRI identifying a datatype that determines how the lexical form maps to a literal value, and
- if and only if the datatype IRI is `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`, a non-empty language tag. The language tag MUST be well-formed according to section 2.2.9 of Tags for identifying languages.

A literal is a language-tagged string if the third element is present. Lexical representations of language tags MAY be converted to lower case. The value space of language tags is always in lower case.

—

For valid XSD datatypes, the lexical form is optionally normalized at construction time. Default behaviour is set by rdflib.NORMALIZE_LITERALS and can be overridden by the normalize parameter to __new__

Equality and hashing of Literals are done based on the lexical form, i.e.:

```
>>> from rdflib.namespace import XSD
```

```
>>> Literal('01') != Literal('1')  # clear - strings differ
True
```

but with data-type they get normalized:

```
>>> Literal('01', datatype=XSD.integer) != Literal('1', datatype=XSD.integer)
False
```

unless disabled:

```
>>> Literal('01', datatype=XSD.integer, normalize=False) != Literal('1',␣
↪datatype=XSD.integer)
True
```

Value based comparison is possible:

```
>>> Literal('01', datatype=XSD.integer).eq(Literal('1', datatype=XSD.float))
True
```

The eq method also provides limited support for basic python types:

```
>>> Literal(1).eq(1) # fine - int compatible with xsd:integer
True
>>> Literal('a').eq('b') # fine - str compatible with plain-lit
False
>>> Literal('a', datatype=XSD.string).eq('a') # fine - str compatible with␣
↪xsd:string
True
>>> Literal('a').eq(1) # not fine, int incompatible with plain-lit
NotImplemented
```

Greater-than/less-than ordering comparisons are also done in value space, when compatible datatypes are used. Incompatible datatypes are ordered by DT, or by lang-tag. For other nodes the ordering is None < BNode < URIRef < Literal

Any comparison with non-rdflib Node are "NotImplemented" In PY3 this is an error.

```
>>> from rdflib import Literal, XSD
>>> lit2006 = Literal('2006-01-01',datatype=XSD.date)
>>> lit2006.toPython()
datetime.date(2006, 1, 1)
>>> lit2006 < Literal('2007-01-01',datatype=XSD.date)
True
>>> Literal(datetime.utcnow()).datatype
rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#dateTime')
>>> Literal(1) > Literal(2) # by value
False
>>> Literal(1) > Literal(2.0) # by value
False
>>> Literal('1') > Literal(1) # by DT
True
>>> Literal('1') < Literal('1') # by lexical form
False
>>> Literal('a', lang='en') > Literal('a', lang='fr') # by lang-tag
```

(continues on next page)

(continued from previous page)

```
False
>>> Literal(1) > URIRef('foo') # by node-type
True
```

The > < operators will eat this NotImplemented and throw a TypeError (py3k):

```
>>> Literal(1).__gt__(2.0)
NotImplemented
```

__abs__()

```
>>> abs(Literal(-1))
rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> from rdflib.namespace import XSD
>>> abs( Literal("-1", datatype=XSD.integer))
rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> abs(Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
```

Return type
Literal

__add__(*val*)

```
>>> from rdflib.namespace import XSD
>>> Literal(1) + 1
rdflib.term.Literal(u'2', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> Literal("1") + "1"
rdflib.term.Literal(u'11')
```

Handling dateTime/date/time based operations in Literals >>> a = Literal('2006-01-01T20:50:00', datatype=XSD.dateTime) >>> b = Literal('P31D', datatype=XSD.duration) >>> (a + b) rdflib.term.Literal('2006-02-01T20:50:00', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dateTime')) >>> from rdflib.namespace import XSD >>> a = Literal('2006-07-01T20:52:00', datatype=XSD.dateTime) >>> b = Literal('P122DT15H58M', datatype=XSD.duration) >>> (a + b) rdflib.term.Literal('2006-11-01T12:50:00', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dateTime'))

Parameters
val (Any) –

Return type
Literal

__annotations__ = {'_datatype': typing.Optional[rdflib.term.URIRef], '_ill_typed': typing.Optional[bool], '_language': typing.Optional[str], '_value': typing.Any}

__bool__()

Is the Literal "True" This is used for if statements, bool(literal), etc.

Return type
bool

__eq__(*other*)

Literals are only equal to other literals.

"Two literals are equal if and only if all of the following hold: * The strings of the two lexical forms compare equal, character by character. * Either both or neither have language tags. * The language tags, if any, compare equal. * Either both or neither have datatype URIs. * The two datatype URIs, if any, compare equal, character by character." – 6.5.1 Literal Equality (RDF: Concepts and Abstract Syntax)

```
>>> Literal("1", datatype=URIRef("foo")) == Literal("1", datatype=URIRef("foo"))
True
>>> Literal("1", datatype=URIRef("foo")) == Literal("1", datatype=URIRef("foo2
↪"))
False
```

```
>>> Literal("1", datatype=URIRef("foo")) == Literal("2", datatype=URIRef("foo"))
False
>>> Literal("1", datatype=URIRef("foo")) == "asdf"
False
>>> from rdflib import XSD
>>> Literal('2007-01-01', datatype=XSD.date) == Literal('2007-01-01',␣
↪datatype=XSD.date)
True
>>> Literal('2007-01-01', datatype=XSD.date) == date(2007, 1, 1)
False
>>> Literal("one", lang="en") == Literal("one", lang="en")
True
>>> Literal("hast", lang='en') == Literal("hast", lang='de')
False
>>> Literal("1", datatype=XSD.integer) == Literal(1)
True
>>> Literal("1", datatype=XSD.integer) == Literal("01", datatype=XSD.integer)
True
```

Parameters
other (Any) –

Return type
bool

__ge__(*other*)

Return self>=value.

Parameters
other (Any) –

Return type
bool

__getstate__()

Return type
Tuple[None, Dict[str, Optional[str]]]

__gt__(*other*)

This implements ordering for Literals, the other comparison methods delegate here

This tries to implement this: http://www.w3.org/TR/sparql11-query/#modOrderBy

In short, Literals with compatible data-types are ordered in value space, i.e. >>> from rdflib import XSD

```
>>> Literal(1) > Literal(2) # int/int
False
>>> Literal(2.0) > Literal(1) # double/int
True
>>> from decimal import Decimal
>>> Literal(Decimal("3.3")) > Literal(2.0) # decimal/double
True
>>> Literal(Decimal("3.3")) < Literal(4.0) # decimal/double
True
>>> Literal('b') > Literal('a') # plain lit/plain lit
True
>>> Literal('b') > Literal('a', datatype=XSD.string) # plain lit/xsd:str
True
```

Incompatible datatype mismatches ordered by DT

```
>>> Literal(1) > Literal("2") # int>string
False
```

Langtagged literals by lang tag >>> Literal("a", lang="en") > Literal("a", lang="fr") False

Parameters
other (Any) –

Return type
bool

__hash__()

```
>>> from rdflib.namespace import XSD
>>> a = {Literal('1', datatype=XSD.integer):'one'}
>>> Literal('1', datatype=XSD.double) in a
False
```

"Called for the key object for dictionary operations, and by the built-in function hash(). Should return a 32-bit integer usable as a hash value for dictionary operations. The only required property is that objects which compare equal have the same hash value; it is advised to somehow mix together (e.g., using exclusive or) the hash values for the components of the object that also play a part in comparison of objects." – 3.4.1 Basic customization (Python)

"Two literals are equal if and only if all of the following hold: * The strings of the two lexical forms compare equal, character by character. * Either both or neither have language tags. * The language tags, if any, compare equal. * Either both or neither have datatype URIs. * The two datatype URIs, if any, compare equal, character by character." – 6.5.1 Literal Equality (RDF: Concepts and Abstract Syntax)

Return type
int

__invert__()

```
>>> ~(Literal(-1))
rdflib.term.Literal(u'0', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> from rdflib.namespace import XSD
>>> ~( Literal("-1", datatype=XSD.integer))
rdflib.term.Literal(u'0', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

Not working:

```
>>> ~(Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
```

Return type
Literal

__le__(*other*)

```
>>> from rdflib.namespace import XSD
>>> Literal('2007-01-01T10:00:00', datatype=XSD.dateTime
...     ) <= Literal('2007-01-01T10:00:00', datatype=XSD.dateTime)
True
```

Parameters
other (Any) –

Return type
bool

__lt__(*other*)

Return self<value.

Parameters
other (Any) –

Return type
bool

__module__ = 'rdflib.term'

__neg__()

```
>>> (- Literal(1))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> (- Literal(10.5))
rdflib.term.Literal(u'-10.5', datatype=rdflib.term.URIRef(u'http://www.w3.org/
↪2001/XMLSchema#double'))
>>> from rdflib.namespace import XSD
```

(continues on next page)

(continued from previous page)

```
>>> (- Literal("1", datatype=XSD.integer))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> (- Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
>>>
```

Return type
Literal

static __new__(*cls*, *lexical_or_value*, *lang=None*, *datatype=None*, *normalize=None*)

Parameters

- **lexical_or_value** (Any) –
- **lang** (Optional[str]) –
- **datatype** (Optional[str]) –
- **normalize** (Optional[bool]) –

Return type
Literal

__pos__()

```
>>> (+ Literal(1))
rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> (+ Literal(-1))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> from rdflib.namespace import XSD
>>> (+ Literal("-1", datatype=XSD.integer))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> (+ Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
```

Return type
Literal

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Literal*], Tuple[str, Optional[str], Optional[str]]]

__repr__()

Return repr(self).

Return type
str

__setstate__(*arg*)

Parameters
arg (Tuple[Any, Dict[str, Any]]) –

Return type
None

__slots__ = ('_language', '_datatype', '_value', '_ill_typed')

__sub__(*val*)

```
>>> from rdflib.namespace import XSD
>>> Literal(2) - 1
rdflib.term.Literal('1', datatype=rdflib.term.URIRef('http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> Literal(1.1) - 1.0
rdflib.term.Literal('0.10000000000000009', datatype=rdflib.term.URIRef('http://
↪www.w3.org/2001/XMLSchema#double'))
>>> Literal(1.1) - 1
rdflib.term.Literal('0.1', datatype=rdflib.term.URIRef('http://www.w3.org/2001/
↪XMLSchema#decimal'))
>>> Literal(1.1, datatype=XSD.float) - Literal(1.0, datatype=XSD.float)
rdflib.term.Literal('0.10000000000000009', datatype=rdflib.term.URIRef('http://
↪www.w3.org/2001/XMLSchema#float'))
>>> Literal("1.1") - 1.0
Traceback (most recent call last):
...
TypeError: Not a number; rdflib.term.Literal('1.1')
>>> Literal(1.1, datatype=XSD.integer) - Literal(1.0, datatype=XSD.integer)
rdflib.term.Literal('0.10000000000000009', datatype=rdflib.term.URIRef('http://
↪www.w3.org/2001/XMLSchema#integer'))
```

Handling dateTime/date/time based operations in Literals >>> a = Literal('2006-01-01T20:50:00', datatype=XSD.dateTime) >>> b = Literal('2006-02-01T20:50:00', datatype=XSD.dateTime) >>> (b - a) rdflib.term.Literal('P31D', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration')) >>> from rdflib.namespace import XSD >>> a = Literal('2006-07-01T20:52:00', datatype=XSD.dateTime) >>> b = Literal('2006-11-01T12:50:00', datatype=XSD.dateTime) >>> (a - b) rdflib.term.Literal('-P122DT15H58M', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration')) >>> (b - a) rdflib.term.Literal('P122DT15H58M', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration'))

Parameters
val (Any) –

Return type
Literal

property datatype: Optional[*URIRef*]

Return type
Optional[*URIRef*]

eq(*other*)

Compare the value of this literal with something else

Either, with the value of another literal comparisons are then done in literal "value space", and according to the rules of XSD subtype-substitution/type-promotion

OR, with a python object:

basestring objects can be compared with plain-literals, or those with datatype xsd:string

bool objects with xsd:boolean

a int, long or float with numeric xsd types

isodate date,time,datetime objects with xsd:date,xsd:time or xsd:datetime

Any other operations returns NotImplemented

Parameters
other (Any) –

Return type
bool

property ill_typed: Optional[bool]

For recognized datatype IRIs, this value will be True if the literal is ill formed, otherwise it will be False. *Literal.value* (i.e. the literal value) should always be defined if this property is False, but should not be considered reliable if this property is True.

If the literal's datatype is None or not in the set of recognized datatype IRIs this value will be None.

Return type
Optional[bool]

property language: Optional[str]

Return type
Optional[str]

n3(*namespace_manager=None*)

Returns a representation in the N3 format.

Examples:

```
>>> Literal("foo").n3()
u'"foo"'
```

Strings with newlines or triple-quotes:

```
>>> Literal("foo\nbar").n3()
u'"""foo\nbar"""'

>>> Literal("'''\'").n3()
u'"\'\'\'"'

>>> Literal('"""').n3()
u'"\\"\\"\\""'
```

Language:

```
>>> Literal("hello", lang="en").n3()
u'"hello"@en'
```

Datatypes:

```
>>> Literal(1).n3()
u'"1"^^<http://www.w3.org/2001/XMLSchema#integer>'

>>> Literal(1.0).n3()
u'"1.0"^^<http://www.w3.org/2001/XMLSchema#double>'

>>> Literal(True).n3()
u'"true"^^<http://www.w3.org/2001/XMLSchema#boolean>'
```

Datatype and language isn't allowed (datatype takes precedence):

```
>>> Literal(1, lang="en").n3()
u'"1"^^<http://www.w3.org/2001/XMLSchema#integer>'
```

Custom datatype:

```
>>> footype = URIRef("http://example.org/ns#foo")
>>> Literal("1", datatype=footype).n3()
u'"1"^^<http://example.org/ns#foo>'
```

Passing a namespace-manager will use it to abbreviate datatype URIs:

```
>>> from rdflib import Graph
>>> Literal(1).n3(Graph().namespace_manager)
u'"1"^^xsd:integer'
```

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

neq(*other*)

A "semantic"/interpreted not equal function, by default, same as __ne__

Parameters
other (Any) –

Return type
bool

normalize()

Returns a new literal with a normalised lexical representation of this literal >>> from rdflib import XSD >>> Literal("01", datatype=XSD.integer, normalize=False).normalize() rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#integer'))

Illegal lexical forms for the datatype given are simply passed on >>> Literal("a", datatype=XSD.integer, normalize=False) rdflib.term.Literal(u'a', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#integer'))

Return type
Literal

toPython()

Returns an appropriate python datatype derived from this RDF Literal

Return type
Any

property value: Any

Return type
Any

class rdflib.term.**Node**

Bases: object

A Node in the Graph.

__module__ = 'rdflib.term'

__slots__ = ()

class rdflib.term.**URIRef**(*value: str*, *base: Optional[str] = None*)

Bases: *IdentifiedNode*

RDF 1.1's IRI Section https://www.w3.org/TR/rdf11-concepts/#section-IRIs

---

Note: Documentation on RDF outside of RDFLib uses the term IRI or URI whereas this class is called URIRef. This is because it was made when the first version of the RDF specification was current, and it used the term *URIRef*, see RDF 1.0 URIRef

---

An IRI (Internationalized Resource Identifier) within an RDF graph is a Unicode string that conforms to the syntax defined in RFC 3987.

IRIs in the RDF abstract syntax MUST be absolute, and MAY contain a fragment identifier.

IRIs are a generalization of URIs [RFC3986] that permits a wider range of Unicode characters.

__add__(*other*)

Return self+value.

Return type
URIRef

__annotations__ = {'__invert__': typing.Callable[[ForwardRef('URIRef')], ForwardRef('InvPath')], '__neg__': typing.Callable[[ForwardRef('URIRef')], ForwardRef('NegatedPath')], '__or__': typing.Callable[[ForwardRef('URIRef'), typing.Union[ForwardRef('URIRef'), ForwardRef('Path')]], ForwardRef('AlternativePath')], '__truediv__': typing.Callable[[ForwardRef('URIRef'), typing.Union[ForwardRef('URIRef'), ForwardRef('Path')]], ForwardRef('SequencePath')]}

__invert__()

inverse path

Parameters
p (Union[*URIRef*, *Path*]) –

Return type
InvPath

__mod__(*other*)

Return self%value.

Return type
URIRef

__module__ = 'rdflib.term'

__mul__(*mul*)

cardinality path

Parameters

- **p** (Union[*URIRef*, *Path*]) –
- **mul** (Literal['*', '+', '?']) –

Return type
MulPath

__neg__()

negated path

Parameters
p (Union[*URIRef*, *AlternativePath*, *InvPath*]) –

Return type
NegatedPath

static __new__(*cls*, *value*, *base=None*)

Parameters

- **value** (str) –
- **base** (Optional[str]) –

Return type
URIRef

__or__(*other*)

alternative path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

__radd__(*other*)

Return type
URIRef

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*URIRef*], Tuple[str]]

__repr__()

Return repr(self).

Return type
str

__slots__ = ()

__truediv__(*other*)

sequence path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

de_skolemize()

Create a Blank Node from a skolem URI, in accordance with http://www.w3.org/TR/rdf11-concepts/#section-skolemization. This function accepts only rdflib type skolemization, to provide a round-tripping within the system.

New in version 4.0.

Return type
BNode

defrag()

Return type
URIRef

property fragment: str

Return the URL Fragment

```
>>> URIRef("http://example.com/some/path/#some-fragment").fragment
'some-fragment'
>>> URIRef("http://example.com/some/path/").fragment
''
```

Return type
str

n3(*namespace_manager=None*)

This will do a limited check for valid URIs, essentially just making sure that the string includes no illegal characters (<, >, ", {, }, |, \, `, ^)

Parameters
namespace_manager (Optional[*NamespaceManager*]) – if not None, will be used to make up a prefixed name

Return type
str

class rdflib.term.**Variable**(*value: str*)

Bases: *Identifier*

A Variable - this is used for querying, or in Formula aware graphs, where Variables can be stored

__module__ = 'rdflib.term'

static __new__(*cls*, *value*)

Parameters
value (str) –

Return type
Variable

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Variable*], Tuple[str]]

__repr__()

Return repr(self).

Return type
str

__slots__ = ()

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

toPython()

Return type
str

rdflib.term.**bind**(*datatype*, *pythontype*, *constructor=None*, *lexicalizer=None*, *datatype_specific=False*)

register a new datatype<->pythontype binding

Parameters

- **constructor** (Optional[Callable[[str], Any]]) – an optional function for converting lexical forms into a Python instances, if not given the pythontype is used directly
- **lexicalizer** (Optional[Callable[[Any], Union[str, bytes]]]) – an optional function for converting python objects to lexical form, if not given object.__str__ is used
- **datatype_specific** (bool) – makes the lexicalizer function be accessible from the pair (pythontype, datatype) if set to True or from the pythontype otherwise. False by default
- **datatype** (str) –
- **pythontype** (Type[Any]) –

Return type
None

rdflib.util module

rdflib.util.**date_time**(*t=None*, *local_time_zone=False*)

http://www.w3.org/TR/NOTE-datetime ex: 1997-07-16T19:20:30Z

```
>>> date_time(1126482850)
'2005-09-11T23:54:10Z'
```

@@ this will change depending on where it is run #>>> date_time(1126482850, local_time_zone=True) #'2005-09-11T19:54:10-04:00'

```
>>> date_time(1)
'1970-01-01T00:00:01Z'
```

```
>>> date_time(0)
'1970-01-01T00:00:00Z'
```

rdflib.util.**find_roots**(*graph*, *prop*, *roots=None*)

Find the roots in some sort of transitive hierarchy.

find_roots(graph, rdflib.RDFS.subClassOf) will return a set of all roots of the sub-class hierarchy

Assumes triple of the form (child, prop, parent), i.e. the direction of RDFS.subClassOf or SKOS.broader

Parameters

- **graph** (*Graph*) –
- **prop** (*URIRef*) –
- **roots** (Optional[Set[*Node*]]) –

Return type

Set[*Node*]

rdflib.util.**first**(*seq*)

return the first element in a python sequence for graphs, use graph.value instead

Parameters

seq (Iterable[TypeVar(_AnyT)]) –

Return type

Optional[TypeVar(_AnyT)]

rdflib.util.**from_n3**(*s*, *default=None*, *backend=None*, *nsm=None*)

Creates the Identifier corresponding to the given n3 string.

```
>>> from rdflib.term import URIRef, Literal
>>> from rdflib.namespace import NamespaceManager
>>> from_n3('<http://ex.com/foo>') == URIRef('http://ex.com/foo')
True
>>> from_n3('"foo"@de') == Literal('foo', lang='de')
True
>>> from_n3('"""multi\nline\nstring"""@en') == Literal(
...     'multi\nline\nstring', lang='en')
True
>>> from_n3('42') == Literal(42)
True
```

(continues on next page)

(continued from previous page)

```
>>> from_n3(Literal(42).n3()) == Literal(42)
True
>>> from_n3('"42"^^xsd:integer') == Literal(42)
True
>>> from rdflib import RDFS
>>> from_n3('rdfs:label') == RDFS['label']
True
>>> nsm = NamespaceManager(rdflib.graph.Graph())
>>> nsm.bind('dbpedia', 'http://dbpedia.org/resource/')
>>> berlin = URIRef('http://dbpedia.org/resource/Berlin')
>>> from_n3('dbpedia:Berlin', nsm=nsm) == berlin
True
```

Parameters

- **s** (str) –
- **default** (Optional[str]) –
- **backend** (Optional[str]) –
- **nsm** (Optional[*NamespaceManager*]) –

Return type
Union[*Node*, str, None]

rdflib.util.**get_tree**(*graph*, *root*, *prop*, *mapper=<function <lambda>>*, *sortkey=None*, *done=None*, *dir='down'*)

Return a nested list/tuple structure representing the tree built by the transitive property given, starting from the root given

i.e.

get_tree(graph,
rdflib.URIRef("http://xmlns.com/foaf/0.1/Person"), rdflib.RDFS.subClassOf)

will return the structure for the subClassTree below person.

dir='down' assumes triple of the form (child, prop, parent), i.e. the direction of RDFS.subClassOf or SKOS.broader Any other dir traverses in the other direction

Parameters

- **graph** (*Graph*) –
- **root** (*Node*) –
- **prop** (*URIRef*) –
- **mapper** (Callable[[*Node*], *Node*]) –
- **sortkey** (Optional[Callable[[Any], Any]]) –
- **done** (Optional[Set[*Node*]]) –
- **dir** (str) –

Return type
Optional[Tuple[*Node*, List[Any]]]

rdflib.util.**guess_format**(*fpath*, *fmap=None*)

Guess RDF serialization based on file suffix. Uses SUFFIX_FORMAT_MAP unless fmap is provided. Examples:

```
>>> guess_format('path/to/file.rdf')
'xml'
>>> guess_format('path/to/file.owl')
'xml'
>>> guess_format('path/to/file.ttl')
'turtle'
>>> guess_format('path/to/file.json')
'json-ld'
>>> guess_format('path/to/file.xhtml')
'rdfa'
>>> guess_format('path/to/file.svg')
'rdfa'
>>> guess_format('path/to/file.xhtml', {'xhtml': 'grddl'})
'grddl'
```

This also works with just the suffixes, with or without leading dot, and regardless of letter case:

```
>>> guess_format('.rdf')
'xml'
>>> guess_format('rdf')
'xml'
>>> guess_format('RDF')
'xml'
```

Parameters

- **fpath** (str) –
- **fmap** (Optional[Dict[str, str]]) –

Return type

Optional[str]

rdflib.util.**list2set**(*seq*)

Return a new list without duplicates. Preserves the order, unlike set(seq)

Parameters

seq (Iterable[TypeVar(_HashableT, bound= Hashable)]) –

Return type

List[TypeVar(_HashableT, bound= Hashable)]

rdflib.util.**more_than**(*sequence*, *number*)

Returns 1 if sequence has more items than number and 0 if not.

Parameters

- **sequence** (Iterable[Any]) –
- **number** (int) –

Return type

int

rdflib.util.**parse_date_time**(*val*)

always returns seconds in UTC

tests are written like this to make any errors easier to understand >>> parse_date_time('2005-09-11T23:54:10Z') - 1126482850.0 0.0

```
>>> parse_date_time('2005-09-11T16:54:10-07:00') - 1126482850.0
0.0
```

```
>>> parse_date_time('1970-01-01T00:00:01Z') - 1.0
0.0
```

```
>>> parse_date_time('1970-01-01T00:00:00Z') - 0.0
0.0
>>> parse_date_time("2005-09-05T10:42:00") - 1125916920.0
0.0
```

Parameters
val (str) –

Return type
int

rdflib.util.**to_term**(*s*, *default=None*)

Creates and returns an Identifier of type corresponding to the pattern of the given positional argument string `s`:

'' returns the `default` keyword argument value or `None`

'<s>' returns `URIRef(s)` (i.e. without angle brackets)

'"s"' returns `Literal(s)` (i.e. without doublequotes)

'_s' returns `BNode(s)` (i.e. without leading underscore)

Parameters

- **s** (Optional[str]) –
- **default** (Optional[*Identifier*]) –

Return type
Optional[*Identifier*]

rdflib.util.**uniq**(*sequence*, *strip=0*)

removes duplicate strings from the sequence.

Parameters

- **sequence** (Iterable[str]) –
- **strip** (int) –

Return type
Set[str]

rdflib.void module

rdflib.void.**generateVoID**(*g*, *dataset=None*, *res=None*, *distinctForPartitions=True*)

Returns a new graph with a VoID description of the passed dataset

For more info on Vocabulary of Interlinked Datasets (VoID), see: http://vocab.deri.ie/void

This only makes two passes through the triples (once to detect the types of things)

The tradeoff is that lots of temporary structures are built up in memory meaning lots of memory may be consumed :) I imagine at least a few copies of your original graph.

the distinctForPartitions parameter controls whether distinctSubjects/objects are tracked for each class/propertyPartition this requires more memory again

Module contents

A pure Python package providing the core RDF constructs.

The packages is intended to provide the core RDF types and interfaces for working with RDF. The package defines a plugin interface for parsers, stores, and serializers that other packages can use to implement parsers, stores, and serializers that will plug into the rdflib package.

The primary interface *rdflib* exposes to work with RDF is *rdflib.graph.Graph*.

A tiny example:

```
>>> from rdflib import Graph, URIRef, Literal
```

```
>>> g = Graph()
>>> result = g.parse("http://www.w3.org/2000/10/swap/test/meet/blue.rdf")
```

```
>>> print("graph has %s statements." % len(g))
graph has 4 statements.
>>>
>>> for s, p, o in g:
...     if (s, p, o) not in g:
...         raise Exception("It better be!")
```

```
>>> s = g.serialize(format='nt')
>>>
>>> sorted(g) == [
...  (URIRef("http://meetings.example.com/cal#m1"),
...   URIRef("http://www.example.org/meeting_organization#homePage"),
...   URIRef("http://meetings.example.com/m1/hp")),
...  (URIRef("http://www.example.org/people#fred"),
...   URIRef("http://www.example.org/meeting_organization#attending"),
...   URIRef("http://meetings.example.com/cal#m1")),
...  (URIRef("http://www.example.org/people#fred"),
...   URIRef("http://www.example.org/personal_details#GivenName"),
...   Literal("Fred")),
...  (URIRef("http://www.example.org/people#fred"),
...   URIRef("http://www.example.org/personal_details#hasEmail"),
...   URIRef("mailto:fred@example.com"))
... ]
True
```

class rdflib.BNode(*value: ~typing.Optional[str] = None*, *_sn_gen: ~typing.Callable[[], str] = <function _serial_number_generator.<locals>._generator>*, *_prefix: str = 'N'*)

Bases: *IdentifiedNode*

RDF 1.1's Blank Nodes Section: https://www.w3.org/TR/rdf11-concepts/#section-blank-nodes

Blank Nodes are local identifiers for unnamed nodes in RDF graphs that are used in some concrete RDF syntaxes or RDF store implementations. They are always locally scoped to the file or RDF store, and are not persistent or portable identifiers for blank nodes. The identifiers for Blank Nodes are not part of the RDF abstract syntax, but are entirely dependent on particular concrete syntax or implementation (such as Turtle, JSON-LD).

—

RDFLib's BNode class makes unique IDs for all the Blank Nodes in a Graph but you should *never* expect, or reply on, BNodes' IDs to match across graphs, or even for multiple copies of the same graph, if they are regenerated from some non-RDFLib source, such as loading from RDF data.

__annotations__ = {}

__module__ = 'rdflib.term'

static __new__(*cls*, *value=None*, *_sn_gen=<function _serial_number_generator.<locals>._generator>*, *_prefix='N'*)

only store implementations should pass in a value

Parameters

- **value** (Optional[str]) –
- **_sn_gen** (Callable[[], str]) –
- **_prefix** (str) –

Return type
BNode

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*BNode*], Tuple[str]]

__repr__()

Return repr(self).

Return type
str

__slots__ = ()

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

skolemize(*authority=None*, *basepath=None*)

Create a URIRef "skolem" representation of the BNode, in accordance with http://www.w3.org/TR/rdf11-concepts/#section-skolemization

New in version 4.0.

Parameters

- **authority** (Optional[str]) –
- **basepath** (Optional[str]) –

Return type
URIRef

class rdflib.**BRICK**

Bases: *DefinedNamespace*

Brick Ontology classes, properties and entity properties. See https://brickschema.org/ for more information.

Generated from: https://github.com/BrickSchema/Brick/releases/download/nightly/Brick.ttl Date: 2021-09-22T14:32:56

AED: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#AED')

AHU: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#AHU')

Ablutions_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ablutions_Room')

Absorption_Chiller: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Absorption_Chiller')

Acceleration_Time_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Acceleration_Time_Setpoint')

Access_Control_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Access_Control_Equipment')

Access_Reader: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Access_Reader')

Active_Chilled_Beam: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Active_Chilled_Beam')

Active_Power_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Active_Power_Sensor')

Adjust_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Adjust_Sensor')

Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air')

Air_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Alarm')

Air_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Differential_Pressure_Sensor')

Air_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Differential_Pressure_Setpoint')

Air_Diffuser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Diffuser')

Air_Enthalpy_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Enthalpy_Sensor')

Air_Flow_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Flow_Deadband_Setpoint')

Air_Flow_Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Flow_Demand_Setpoint')

Air_Flow_Loss_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Flow_Loss_Alarm')

Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Flow_Sensor')

Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Flow_Setpoint')

Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Flow_Setpoint_Limit')

Air_Grains_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Grains_Sensor')

Air_Handler_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Handler_Unit')

Air_Handling_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Handling_Unit')

Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Humidity_Setpoint')

Air_Loop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Loop')

Air_Plenum: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Plenum')

Air_Quality_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Quality_Sensor')

Air_Static_Pressure_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Static_Pressure_Step_Parameter')

Air_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_System')

Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Temperature_Alarm')

Air_Temperature_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Temperature_Integral_Time_Parameter')

Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Temperature_Sensor')

Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Temperature_Setpoint')

Air_Temperature_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Temperature_Setpoint_Limit')

Air_Temperature_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Temperature_Step_Parameter')

Air_Wet_Bulb_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Air_Wet_Bulb_Temperature_Sensor')

Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Alarm')

Alarm_Delay_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Alarm_Delay_Parameter')

Angle_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Angle_Sensor')

Auditorium: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Auditorium')

Automated_External_Defibrillator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Automated_External_Defibrillator')

Automatic_Mode_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Automatic_Mode_Command')

Availability_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Availability_Status')

Average_Cooling_Demand_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Average_Cooling_Demand_Sensor')

Average_Discharge_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Average_Discharge_Air_Flow_Sensor')

Average_Exhaust_Air_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Average_Exhaust_Air_Static_Pressure_Sensor')

Average_Heating_Demand_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Average_Heating_Demand_Sensor')

Average_Supply_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Average_Supply_Air_Flow_Sensor')

Average_Zone_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Average_Zone_Air_Temperature_Sensor')

Baseboard_Radiator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Baseboard_Radiator')

Basement: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Basement')

Battery: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Battery')

Battery_Energy_Storage_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Battery_Energy_Storage_System')

Battery_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Battery_Room')

Battery_Voltage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Battery_Voltage_Sensor')

Bench_Space: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bench_Space')

Blowdown_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Blowdown_Water')

Boiler: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Boiler')

Booster_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Booster_Fan')

Box_Mode_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Box_Mode_Command')

Break_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Break_Room')

Breaker_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Breaker_Panel')

Breakroom: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Breakroom')

Broadcast_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Broadcast_Room')

Building: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building')

Building_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Air')

Building_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Air_Humidity_Setpoint')

Building_Air_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Air_Static_Pressure_Sensor')

Building_Air_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Air_Static_Pressure_Setpoint')

Building_Chilled_Water_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Chilled_Water_Meter')

Building_Electrical_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Electrical_Meter')

Building_Gas_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Gas_Meter')

Building_Hot_Water_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Hot_Water_Meter')

Building_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Meter')

Building_Water_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Building_Water_Meter')

Bus_Riser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bus_Riser')

Bypass_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Air')

Bypass_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Air_Flow_Sensor')

Bypass_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Air_Humidity_Setpoint')

Bypass_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Command')

Bypass_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Valve')

Bypass_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Water')

Bypass_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Water_Flow_Sensor')

Bypass_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Bypass_Water_Flow_Setpoint')

CAV: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CAV')

CO: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO')

CO2: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO2')

CO2_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO2_Alarm')

CO2_Differential_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO2_Differential_Sensor')

CO2_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO2_Level_Sensor')

CO2_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO2_Sensor')

CO2_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO2_Setpoint')

CO_Differential_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO_Differential_Sensor')

CO_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO_Level_Sensor')

CO_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CO_Sensor')

CRAC: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#CRAC')

Cafeteria: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cafeteria')

Camera: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Camera')

Capacity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Capacity_Sensor')

Ceiling_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ceiling_Fan')

Centrifugal_Chiller: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Centrifugal_Chiller')

Change_Filter_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Change_Filter_Alarm')

Chilled_Beam: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Beam')

Chilled_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water')

Chilled_Water_Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Coil')

Chilled_Water_Differential_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Deadband_Setpoint')

Chilled_Water_Differential_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Integral_Time_Parameter')

Chilled_Water_Differential_Pressure_Load_Shed_Reset_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Load_Shed_Reset_Status')

Chilled_Water_Differential_Pressure_Load_Shed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Load_Shed_Setpoint')

Chilled_Water_Differential_Pressure_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Load_Shed_Status')

Chilled_Water_Differential_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Proportional_Band_Parameter')

Chilled_Water_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Sensor')

Chilled_Water_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Setpoint')

Chilled_Water_Differential_Pressure_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Pressure_Step_Parameter')

Chilled_Water_Differential_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Differential_Temperature_Sensor')

Chilled_Water_Discharge_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Discharge_Flow_Sensor')

Chilled_Water_Discharge_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Discharge_Flow_Setpoint')

Chilled_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Flow_Sensor')

Chilled_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Flow_Setpoint')

Chilled_Water_Loop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Loop')

Chilled_Water_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Meter')

Chilled_Water_Pump: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Pump')

Chilled_Water_Pump_Differential_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Pump_Differential_Pressure_Deadband_Setpoint')

Chilled_Water_Return_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Return_Flow_Sensor')

Chilled_Water_Return_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Return_Temperature_Sensor')

Chilled_Water_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Static_Pressure_Setpoint')

Chilled_Water_Supply_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Supply_Flow_Sensor')

Chilled_Water_Supply_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Supply_Flow_Setpoint')

Chilled_Water_Supply_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Supply_Temperature_Sensor')

Chilled_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_System')

Chilled_Water_System_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_System_Enable_Command')

Chilled_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Temperature_Sensor')

Chilled_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Temperature_Setpoint')

Chilled_Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chilled_Water_Valve')

Chiller: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Chiller')

Class: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Class')

Close_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Close_Limit')

Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Coil')

Cold_Box: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cold_Box')

Coldest_Zone_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Coldest_Zone_Air_Temperature_Sensor')

Collection: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Collection')

Collection_Basin_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Collection_Basin_Water')

Collection_Basin_Water_Heater: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Collection_Basin_Water_Heater')

Collection_Basin_Water_Level_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Collection_Basin_Water_Level_Alarm')

Collection_Basin_Water_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Collection_Basin_Water_Level_Sensor')

Collection_Basin_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Collection_Basin_Water_Temperature_Sensor')

Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Command')

Common_Space: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Common_Space')

Communication_Loss_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Communication_Loss_Alarm')

Compressor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Compressor')

Computer_Room_Air_Conditioning: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Computer_Room_Air_Conditioning')

Concession: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Concession')

Condensate_Leak_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condensate_Leak_Alarm')

Condenser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser')

Condenser_Heat_Exchanger: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Heat_Exchanger')

Condenser_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water')

Condenser_Water_Bypass_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water_Bypass_Valve')

Condenser_Water_Isolation_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water_Isolation_Valve')

Condenser_Water_Pump: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water_Pump')

Condenser_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water_System')

Condenser_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water_Temperature_Sensor')

Condenser_Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condenser_Water_Valve')

Condensing_Natural_Gas_Boiler: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Condensing_Natural_Gas_Boiler')

Conductivity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Conductivity_Sensor')

Conference_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Conference_Room')

Constant_Air_Volume_Box: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Constant_Air_Volume_Box')

Contact_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Contact_Sensor')

Control_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Control_Room')

Cooling_Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Coil')

Cooling_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Command')

Cooling_Demand_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Demand_Sensor')

Cooling_Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Demand_Setpoint')

Cooling_Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Discharge_Air_Flow_Setpoint')

Cooling_Discharge_Air_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Discharge_Air_Temperature_Deadband_Setpoint')

Cooling_Discharge_Air_Temperature_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Discharge_Air_Temperature_Integral_Time_Parameter')

Cooling_Discharge_Air_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Discharge_Air_Temperature_Proportional_Band_Parameter')

Cooling_Start_Stop_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Start_Stop_Status')

Cooling_Supply_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Supply_Air_Flow_Setpoint')

Cooling_Supply_Air_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Supply_Air_Temperature_Deadband_Setpoint')

Cooling_Supply_Air_Temperature_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Supply_Air_Temperature_Integral_Time_Parameter')

Cooling_Supply_Air_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Supply_Air_Temperature_Proportional_Band_Parameter')

Cooling_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Temperature_Setpoint')

Cooling_Tower: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Tower')

Cooling_Tower_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Tower_Fan')

Cooling_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cooling_Valve')

Copy_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Copy_Room')

Core_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Core_Temperature_Sensor')

Core_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Core_Temperature_Setpoint')

Cubicle: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cubicle')

Current_Imbalance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Current_Imbalance_Sensor')

Current_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Current_Limit')

Current_Output_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Current_Output_Sensor')

Current_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Current_Sensor')

Curtailment_Override_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Curtailment_Override_Command')

Cycle_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Cycle_Alarm')

DC_Bus_Voltage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#DC_Bus_Voltage_Sensor')

DOAS: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#DOAS')

Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Damper')

Damper_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Damper_Command')

Damper_Position_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Damper_Position_Command')

Damper_Position_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Damper_Position_Sensor')

Damper_Position_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Damper_Position_Setpoint')

Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Deadband_Setpoint')

Deceleration_Time_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Deceleration_Time_Setpoint')

Dedicated_Outdoor_Air_System_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Dedicated_Outdoor_Air_System_Unit')

Dehumidification_Start_Stop_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Dehumidification_Start_Stop_Status')

Deionised_Water_Conductivity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Deionised_Water_Conductivity_Sensor')

Deionised_Water_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Deionised_Water_Level_Sensor')

Deionized_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Deionized_Water')

Deionized_Water_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Deionized_Water_Alarm')

Delay_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Delay_Parameter')

Demand_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Demand_Sensor')

Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Demand_Setpoint')

Derivative_Gain_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Derivative_Gain_Parameter')

Derivative_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Derivative_Time_Parameter')

Detention_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Detention_Room')

Dew_Point_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Dew_Point_Setpoint')

Dewpoint_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Dewpoint_Sensor')

Differential_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Air_Temperature_Setpoint')

Differential_Pressure_Bypass_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Bypass_Valve')

Differential_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Deadband_Setpoint')

Differential_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Integral_Time_Parameter')

Differential_Pressure_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Load_Shed_Status')

Differential_Pressure_Proportional_Band: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Proportional_Band')

Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Sensor')

Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Setpoint')

Differential_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Setpoint_Limit')

Differential_Pressure_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Pressure_Step_Parameter')

Differential_Speed_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Speed_Sensor')

Differential_Speed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Speed_Setpoint')

Differential_Supply_Return_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Differential_Supply_Return_Water_Temperature_Sensor')

Dimmer: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Dimmer')

Direct_Expansion_Cooling_Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Direct_Expansion_Cooling_Coil')

Direct_Expansion_Heating_Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Direct_Expansion_Heating_Coil')

Direction_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Direction_Command')

Direction_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Direction_Sensor')

Direction_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Direction_Status')

Disable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Command')

Disable_Differential_Enthalpy_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Differential_Enthalpy_Command')

Disable_Differential_Temperature_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Differential_Temperature_Command')

Disable_Fixed_Enthalpy_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Fixed_Enthalpy_Command')

Disable_Fixed_Temperature_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Fixed_Temperature_Command')

Disable_Hot_Water_System_Outside_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Hot_Water_System_Outside_Air_Temperature_Setpoint')

Disable_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disable_Status')

Discharge_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air')

Discharge_Air_Dewpoint_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Dewpoint_Sensor')

Discharge_Air_Duct_Pressure_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Duct_Pressure_Status')

Discharge_Air_Flow_Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Flow_Demand_Setpoint')

Discharge_Air_Flow_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Flow_High_Reset_Setpoint')

Discharge_Air_Flow_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Flow_Low_Reset_Setpoint')

Discharge_Air_Flow_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Flow_Reset_Setpoint')

Discharge_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Flow_Sensor')

Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Flow_Setpoint')

Discharge_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Humidity_Sensor')

Discharge_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Humidity_Setpoint')

Discharge_Air_Smoke_Detection_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Smoke_Detection_Alarm')

Discharge_Air_Static_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Static_Pressure_Deadband_Setpoint')

Discharge_Air_Static_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Static_Pressure_Integral_Time_Parameter')

Discharge_Air_Static_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Static_Pressure_Proportional_Band_Parameter')

Discharge_Air_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Static_Pressure_Sensor')

Discharge_Air_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Static_Pressure_Setpoint')

Discharge_Air_Static_Pressure_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Static_Pressure_Step_Parameter')

Discharge_Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Alarm')

Discharge_Air_Temperature_Cooling_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Cooling_Setpoint')

Discharge_Air_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Deadband_Setpoint')

Discharge_Air_Temperature_Heating_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Heating_Setpoint')

Discharge_Air_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_High_Reset_Setpoint')

Discharge_Air_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Low_Reset_Setpoint')

Discharge_Air_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Proportional_Band_Parameter')

Discharge_Air_Temperature_Reset_Differential_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Reset_Differential_Setpoint')

Discharge_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Sensor')

Discharge_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Setpoint')

Discharge_Air_Temperature_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Setpoint_Limit')

Discharge_Air_Temperature_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Temperature_Step_Parameter')

Discharge_Air_Velocity_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Air_Velocity_Pressure_Sensor')

Discharge_Chilled_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Chilled_Water')

Discharge_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Fan')

Discharge_Hot_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Hot_Water')

Discharge_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water')

Discharge_Water_Differential_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Differential_Pressure_Deadband_Setpoint')

Discharge_Water_Differential_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Differential_Pressure_Integral_Time_Parameter')

Discharge_Water_Differential_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Differential_Pressure_Proportional_Band_Parameter')

Discharge_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Flow_Sensor')

Discharge_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Flow_Setpoint')

Discharge_Water_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Temperature_Alarm')

Discharge_Water_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Temperature_Proportional_Band_Parameter')

Discharge_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Temperature_Sensor')

Discharge_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Discharge_Water_Temperature_Setpoint')

Disconnect_Switch: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Disconnect_Switch')

Displacement_Flow_Air_Diffuser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Displacement_Flow_Air_Diffuser')

Distribution_Frame: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Distribution_Frame')

Domestic_Hot_Water_Supply_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Hot_Water_Supply_Temperature_Sensor')

Domestic_Hot_Water_Supply_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Hot_Water_Supply_Temperature_Setpoint')

Domestic_Hot_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Hot_Water_System')

Domestic_Hot_Water_System_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Hot_Water_System_Enable_Command')

Domestic_Hot_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Hot_Water_Temperature_Setpoint')

Domestic_Hot_Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Hot_Water_Valve')

Domestic_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Water')

Domestic_Water_Loop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Domestic_Water_Loop')

Drench_Hose: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Drench_Hose')

Drive_Ready_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Drive_Ready_Status')

Duration_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Duration_Sensor')

ESS_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#ESS_Panel')

EconCycle_Start_Stop_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#EconCycle_Start_Stop_Status')

Economizer: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Economizer')

Economizer_Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Economizer_Damper')

Effective_Air_Temperature_Cooling_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Air_Temperature_Cooling_Setpoint')

Effective_Air_Temperature_Heating_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Air_Temperature_Heating_Setpoint')

Effective_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Air_Temperature_Setpoint')

Effective_Discharge_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Discharge_Air_Temperature_Setpoint')

Effective_Return_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Return_Air_Temperature_Setpoint')

Effective_Room_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Room_Air_Temperature_Setpoint')

Effective_Supply_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Supply_Air_Temperature_Setpoint')

Effective_Zone_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Effective_Zone_Air_Temperature_Setpoint')

Electric_Baseboard_Radiator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electric_Baseboard_Radiator')

Electric_Boiler: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electric_Boiler')

Electric_Radiator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electric_Radiator')

Electrical_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electrical_Equipment')

Electrical_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electrical_Meter')

Electrical_Power_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electrical_Power_Sensor')

Electrical_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electrical_Room')

Electrical_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Electrical_System')

Elevator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Elevator')

Elevator_Shaft: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Elevator_Shaft')

Elevator_Space: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Elevator_Space')

Embedded_Surface_System_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Embedded_Surface_System_Panel')

Embedded_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Embedded_Temperature_Sensor')

Embedded_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Embedded_Temperature_Setpoint')

Emergency_Air_Flow_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Air_Flow_System')

Emergency_Air_Flow_System_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Air_Flow_System_Status')

Emergency_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Alarm')

Emergency_Generator_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Generator_Alarm')

Emergency_Generator_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Generator_Status')

Emergency_Phone: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Phone')

Emergency_Power_Off_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Power_Off_System')

Emergency_Power_Off_System_Activated_By_High_Temperature_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Power_Off_System_Activated_By_High_Temperature_Status')

Emergency_Power_Off_System_Activated_By_Leak_Detection_System_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Power_Off_System_Activated_By_Leak_Detection_System_Status')

Emergency_Power_Off_System_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Power_Off_System_Status')

Emergency_Push_Button_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Push_Button_Status')

Emergency_Wash_Station: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Emergency_Wash_Station')

Employee_Entrance_Lobby: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Employee_Entrance_Lobby')

Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Command')

Enable_Differential_Enthalpy_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Differential_Enthalpy_Command')

Enable_Differential_Temperature_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Differential_Temperature_Command')

Enable_Fixed_Enthalpy_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Fixed_Enthalpy_Command')

Enable_Fixed_Temperature_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Fixed_Temperature_Command')

Enable_Hot_Water_System_Outside_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Hot_Water_System_Outside_Air_Temperature_Setpoint')

Enable_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enable_Status')

Enclosed_Office: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enclosed_Office')

Energy_Generation_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_Generation_System')

Energy_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_Sensor')

Energy_Storage: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_Storage')

Energy_Storage_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_Storage_System')

Energy_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_System')

Energy_Usage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_Usage_Sensor')

Energy_Zone: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Energy_Zone')

Entering_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Entering_Water')

Entering_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Entering_Water_Flow_Sensor')

Entering_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Entering_Water_Flow_Setpoint')

Entering_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Entering_Water_Temperature_Sensor')

Entering_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Entering_Water_Temperature_Setpoint')

Enthalpy_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enthalpy_Sensor')

Enthalpy_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Enthalpy_Setpoint')

Entrance: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Entrance')

Environment_Box: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Environment_Box')

Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Equipment')

Equipment_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Equipment_Room')

Evaporative_Heat_Exchanger: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Evaporative_Heat_Exchanger')

Even_Month_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Even_Month_Status')

Exercise_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exercise_Room')

Exhaust_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air')

Exhaust_Air_Dewpoint_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Dewpoint_Sensor')

Exhaust_Air_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Differential_Pressure_Sensor')

Exhaust_Air_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Differential_Pressure_Setpoint')

Exhaust_Air_Flow_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Flow_Integral_Time_Parameter')

Exhaust_Air_Flow_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Flow_Proportional_Band_Parameter')

Exhaust_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Flow_Sensor')

Exhaust_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Flow_Setpoint')

Exhaust_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Humidity_Sensor')

Exhaust_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Humidity_Setpoint')

Exhaust_Air_Stack_Flow_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Stack_Flow_Deadband_Setpoint')

Exhaust_Air_Stack_Flow_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Stack_Flow_Integral_Time_Parameter')

Exhaust_Air_Stack_Flow_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Stack_Flow_Proportional_Band_Parameter')

Exhaust_Air_Stack_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Stack_Flow_Sensor')

Exhaust_Air_Stack_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Stack_Flow_Setpoint')

Exhaust_Air_Static_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Static_Pressure_Proportional_Band_Parameter')

Exhaust_Air_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Static_Pressure_Sensor')

Exhaust_Air_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Static_Pressure_Setpoint')

Exhaust_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Temperature_Sensor')

Exhaust_Air_Velocity_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Air_Velocity_Pressure_Sensor')

Exhaust_Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Damper')

Exhaust_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Fan')

Exhaust_Fan_Disable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Fan_Disable_Command')

Exhaust_Fan_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Exhaust_Fan_Enable_Command')

Eye_Wash_Station: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Eye_Wash_Station')

FCU: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#FCU')

Failure_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Failure_Alarm')

Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fan')

Fan_Coil_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fan_Coil_Unit')

Fan_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fan_On_Off_Status')

Fan_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fan_Status')

Fan_VFD: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fan_VFD')

Fault_Reset_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fault_Reset_Command')

Fault_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fault_Status')

Field_Of_Play: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Field_Of_Play')

Filter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Filter')

Filter_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Filter_Differential_Pressure_Sensor')

Filter_Reset_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Filter_Reset_Command')

Filter_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Filter_Status')

Final_Filter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Final_Filter')

Fire_Control_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fire_Control_Panel')

Fire_Safety_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fire_Safety_Equipment')

Fire_Safety_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fire_Safety_System')

Fire_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fire_Sensor')

Fire_Zone: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fire_Zone')

First_Aid_Kit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#First_Aid_Kit')

First_Aid_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#First_Aid_Room')

Floor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Floor')

Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Flow_Sensor')

Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Flow_Setpoint')

Fluid: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fluid')

Food_Service_Room: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Food_Service_Room')

Formaldehyde_Level_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Formaldehyde_Level_Sensor')

Freeze_Status: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Freeze_Status')

Freezer: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Freezer')

Frequency_Command: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Frequency_Command')

Frequency_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Frequency_Sensor')

Fresh_Air_Fan: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fresh_Air_Fan')

Fresh_Air_Setpoint_Limit: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fresh_Air_Setpoint_Limit')

Frost: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Frost')

Frost_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Frost_Sensor')

Fuel_Oil: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fuel_Oil')

Fume_Hood: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fume_Hood')

Fume_Hood_Air_Flow_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Fume_Hood_Air_Flow_Sensor')

Furniture: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Furniture')

Gain_Parameter: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gain_Parameter')

Gas: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gas')

Gas_Distribution: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gas_Distribution')

Gas_Meter: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gas_Meter')

Gas_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gas_Sensor')

Gas_System: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gas_System')

Gas_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gas_Valve')

Gasoline: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gasoline')

Gatehouse: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Gatehouse')

Generator_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Generator_Room')

Glycol: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Glycol')

HVAC_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#HVAC_Equipment')

HVAC_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#HVAC_System')

HVAC_Zone: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#HVAC_Zone')

HX: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#HX')

Hail: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hail')

Hail_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hail_Sensor')

Hallway: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hallway')

Hazardous_Materials_Storage: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hazardous_Materials_Storage')

Heat_Exchanger: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Exchanger')

Heat_Exchanger_Supply_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Exchanger_Supply_Water_Temperature_Sensor')

Heat_Exchanger_System_Enable_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Exchanger_System_Enable_Status')

Heat_Recovery_Hot_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Recovery_Hot_Water_System')

Heat_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Sensor')

Heat_Wheel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Wheel')

Heat_Wheel_VFD: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heat_Wheel_VFD')

Heating_Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Coil')

Heating_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Command')

Heating_Demand_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Demand_Sensor')

Heating_Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Demand_Setpoint')

Heating_Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Discharge_Air_Flow_Setpoint')

Heating_Discharge_Air_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Discharge_Air_Temperature_Deadband_Setpoint')

Heating_Discharge_Air_Temperature_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Discharge_Air_Temperature_Integral_Time_Parameter')

Heating_Discharge_Air_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Discharge_Air_Temperature_Proportional_Band_Parameter')

Heating_Start_Stop_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Start_Stop_Status')

Heating_Supply_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Supply_Air_Flow_Setpoint')

Heating_Supply_Air_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Supply_Air_Temperature_Deadband_Setpoint')

Heating_Supply_Air_Temperature_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Supply_Air_Temperature_Integral_Time_Parameter')

Heating_Supply_Air_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Supply_Air_Temperature_Proportional_Band_Parameter')

Heating_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Temperature_Setpoint')

Heating_Thermal_Power_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Thermal_Power_Sensor')

Heating_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Valve')

Heating_Ventilation_Air_Conditioning_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Heating_Ventilation_Air_Conditioning_System')

High_CO2_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_CO2_Alarm')

High_Discharge_Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Discharge_Air_Temperature_Alarm')

High_Head_Pressure_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Head_Pressure_Alarm')

High_Humidity_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Humidity_Alarm')

High_Humidity_Alarm_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Humidity_Alarm_Parameter')

High_Outside_Air_Lockout_Temperature_Differential_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Outside_Air_Lockout_Temperature_Differential_Parameter')

High_Return_Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Return_Air_Temperature_Alarm')

High_Static_Pressure_Cutout_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Static_Pressure_Cutout_Setpoint_Limit')

High_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Temperature_Alarm')

High_Temperature_Alarm_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Temperature_Alarm_Parameter')

High_Temperature_Hot_Water_Return_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Temperature_Hot_Water_Return_Temperature_Sensor')

High_Temperature_Hot_Water_Supply_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#High_Temperature_Hot_Water_Supply_Temperature_Sensor')

Hold_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hold_Status')

Hospitality_Box: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hospitality_Box')

Hot_Box: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Box')

Hot_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water')

Hot_Water_Baseboard_Radiator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Baseboard_Radiator')

Hot_Water_Coil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Coil')

Hot_Water_Differential_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Deadband_Setpoint')

Hot_Water_Differential_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Integral_Time_Parameter')

Hot_Water_Differential_Pressure_Load_Shed_Reset_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Load_Shed_Reset_Status')

Hot_Water_Differential_Pressure_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Load_Shed_Status')

Hot_Water_Differential_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Proportional_Band_Parameter')

Hot_Water_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Sensor')

Hot_Water_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Pressure_Setpoint')

Hot_Water_Differential_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Differential_Temperature_Sensor')

Hot_Water_Discharge_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Discharge_Flow_Sensor')

Hot_Water_Discharge_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Discharge_Flow_Setpoint')

Hot_Water_Discharge_Temperature_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Discharge_Temperature_Load_Shed_Status')

Hot_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Flow_Sensor')

Hot_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Flow_Setpoint')

Hot_Water_Loop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Loop')

Hot_Water_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Meter')

Hot_Water_Pump: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Pump')

Hot_Water_Radiator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Radiator')

Hot_Water_Return_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Return_Flow_Sensor')

Hot_Water_Return_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Return_Temperature_Sensor')

Hot_Water_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Static_Pressure_Setpoint')

Hot_Water_Supply_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Supply_Flow_Sensor')

Hot_Water_Supply_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Supply_Flow_Setpoint')

Hot_Water_Supply_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Supply_Temperature_High_Reset_Setpoint')

Hot_Water_Supply_Temperature_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Supply_Temperature_Load_Shed_Status')

Hot_Water_Supply_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Supply_Temperature_Low_Reset_Setpoint')

Hot_Water_Supply_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Supply_Temperature_Sensor')

Hot_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_System')

Hot_Water_System_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_System_Enable_Command')

Hot_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Temperature_Setpoint')

Hot_Water_Usage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Usage_Sensor')

Hot_Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Hot_Water_Valve')

Humidification_Start_Stop_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidification_Start_Stop_Status')

Humidifier: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidifier')

Humidifier_Fault_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidifier_Fault_Status')

Humidify_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidify_Command')

Humidity_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidity_Alarm')

Humidity_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidity_Parameter')

Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidity_Sensor')

Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidity_Setpoint')

Humidity_Tolerance_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Humidity_Tolerance_Parameter')

IDF: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#IDF')

Ice: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ice')

Ice_Tank_Leaving_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ice_Tank_Leaving_Water_Temperature_Sensor')

Illuminance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Illuminance_Sensor')

Imbalance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Imbalance_Sensor')

Induction_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Induction_Unit')

Information_Area: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Information_Area')

Inside_Face_Surface_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Inside_Face_Surface_Temperature_Sensor')

Inside_Face_Surface_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Inside_Face_Surface_Temperature_Setpoint')

Intake_Air_Filter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Intake_Air_Filter')

Intake_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Intake_Air_Temperature_Sensor')

Integral_Gain_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Integral_Gain_Parameter')

Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Integral_Time_Parameter')

Intercom_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Intercom_Equipment')

Interface: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Interface')

Intrusion_Detection_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Intrusion_Detection_Equipment')

Inverter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Inverter')

Isolation_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Isolation_Valve')

Janitor_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Janitor_Room')

Jet_Nozzle_Air_Diffuser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Jet_Nozzle_Air_Diffuser')

Laboratory: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Laboratory')

Laminar_Flow_Air_Diffuser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Laminar_Flow_Air_Diffuser')

Last_Fault_Code_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Last_Fault_Code_Status')

Lead_Lag_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lead_Lag_Command')

Lead_Lag_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lead_Lag_Status')

Lead_On_Off_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lead_On_Off_Command')

Leak_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Leak_Alarm')

Leaving_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Leaving_Water')

Leaving_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Leaving_Water_Flow_Sensor')

Leaving_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Leaving_Water_Flow_Setpoint')

Leaving_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Leaving_Water_Temperature_Sensor')

Leaving_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Leaving_Water_Temperature_Setpoint')

Library: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Library')

Lighting: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lighting')

Lighting_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lighting_Equipment')

Lighting_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lighting_System')

Lighting_Zone: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lighting_Zone')

Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Limit')

Liquid: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Liquid')

Liquid_CO2: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Liquid_CO2')

Liquid_Detection_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Liquid_Detection_Alarm')

Load_Current_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Current_Sensor')

Load_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Parameter')

Load_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Setpoint')

Load_Shed_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Shed_Command')

Load_Shed_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Shed_Differential_Pressure_Setpoint')

Load_Shed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Shed_Setpoint')

Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Load_Shed_Status')

Loading_Dock: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Loading_Dock')

Lobby: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lobby')

Locally_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Locally_On_Off_Status')

Location: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Location')

Lockout_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lockout_Status')

Lockout_Temperature_Differential_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lockout_Temperature_Differential_Parameter')

Loop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Loop')

Lounge: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lounge')

Louver: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Louver')

Low_Freeze_Protect_Temperature_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Freeze_Protect_Temperature_Parameter')

Low_Humidity_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Humidity_Alarm')

Low_Humidity_Alarm_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Humidity_Alarm_Parameter')

Low_Outside_Air_Lockout_Temperature_Differential_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Outside_Air_Lockout_Temperature_Differential_Parameter')

Low_Outside_Air_Temperature_Enable_Differential_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Outside_Air_Temperature_Enable_Differential_Sensor')

Low_Outside_Air_Temperature_Enable_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Outside_Air_Temperature_Enable_Setpoint')

Low_Return_Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Return_Air_Temperature_Alarm')

Low_Suction_Pressure_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Suction_Pressure_Alarm')

Low_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Temperature_Alarm')

Low_Temperature_Alarm_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Low_Temperature_Alarm_Parameter')

Lowest_Exhaust_Air_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Lowest_Exhaust_Air_Static_Pressure_Sensor')

Luminaire: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Luminaire')

Luminaire_Driver: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Luminaire_Driver')

Luminance_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Luminance_Alarm')

Luminance_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Luminance_Command')

Luminance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Luminance_Sensor')

Luminance_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Luminance_Setpoint')

MAU: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#MAU')

MDF: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#MDF')

Mail_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mail_Room')

Maintenance_Mode_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Maintenance_Mode_Command')

Maintenance_Required_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Maintenance_Required_Alarm')

Majlis: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Majlis')

Makeup_Air_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Makeup_Air_Unit')

Makeup_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Makeup_Water')

Makeup_Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Makeup_Water_Valve')

Manual_Auto_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Manual_Auto_Status')

Massage_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Massage_Room')

Max_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Air_Flow_Setpoint_Limit')

Max_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Air_Temperature_Setpoint')

Max_Chilled_Water_Differential_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Chilled_Water_Differential_Pressure_Setpoint_Limit')

Max_Cooling_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Cooling_Discharge_Air_Flow_Setpoint_Limit')

Max_Cooling_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Cooling_Supply_Air_Flow_Setpoint_Limit')

Max_Discharge_Air_Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Discharge_Air_Static_Pressure_Setpoint_Limit')

Max_Discharge_Air_Temperature_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Discharge_Air_Temperature_Setpoint_Limit')

Max_Frequency_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Frequency_Command')

Max_Heating_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Heating_Discharge_Air_Flow_Setpoint_Limit')

Max_Heating_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Heating_Supply_Air_Flow_Setpoint_Limit')

Max_Hot_Water_Differential_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Hot_Water_Differential_Pressure_Setpoint_Limit')

Max_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Limit')

Max_Load_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Load_Setpoint')

Max_Occupied_Cooling_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Occupied_Cooling_Discharge_Air_Flow_Setpoint_Limit')

Max_Occupied_Cooling_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Occupied_Cooling_Supply_Air_Flow_Setpoint_Limit')

Max_Occupied_Heating_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Occupied_Heating_Discharge_Air_Flow_Setpoint_Limit')

Max_Occupied_Heating_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Occupied_Heating_Supply_Air_Flow_Setpoint_Limit')

Max_Position_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Position_Setpoint_Limit')

Max_Speed_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Speed_Setpoint_Limit')

Max_Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Static_Pressure_Setpoint_Limit')

Max_Supply_Air_Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Supply_Air_Static_Pressure_Setpoint_Limit')

Max_Temperature_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Temperature_Setpoint_Limit')

Max_Unoccupied_Cooling_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Unoccupied_Cooling_Discharge_Air_Flow_Setpoint_Limit')

Max_Unoccupied_Cooling_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Unoccupied_Cooling_Supply_Air_Flow_Setpoint_Limit')

Max_Unoccupied_Heating_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Unoccupied_Heating_Discharge_Air_Flow_Setpoint_Limit')

Max_Unoccupied_Heating_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Unoccupied_Heating_Supply_Air_Flow_Setpoint_Limit')

Max_Water_Level_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Water_Level_Alarm')

Max_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Max_Water_Temperature_Setpoint')

Measurable: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Measurable')

Mechanical_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mechanical_Room')

Media_Hot_Desk: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Media_Hot_Desk')

Media_Production_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Media_Production_Room')

Media_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Media_Room')

Medical_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medical_Room')

Medium_Temperature_Hot_Water_Differential_Pressure_Load_Shed_Reset_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Differential_Pressure_Load_Shed_Reset_Status')

Medium_Temperature_Hot_Water_Differential_Pressure_Load_Shed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Differential_Pressure_Load_Shed_Setpoint')

Medium_Temperature_Hot_Water_Differential_Pressure_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Differential_Pressure_Load_Shed_Status')

Medium_Temperature_Hot_Water_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Differential_Pressure_Sensor')

Medium_Temperature_Hot_Water_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Differential_Pressure_Setpoint')

Medium_Temperature_Hot_Water_Discharge_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Discharge_Temperature_High_Reset_Setpoint')

Medium_Temperature_Hot_Water_Discharge_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Discharge_Temperature_Low_Reset_Setpoint')

Medium_Temperature_Hot_Water_Return_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Return_Temperature_Sensor')

Medium_Temperature_Hot_Water_Supply_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Supply_Temperature_High_Reset_Setpoint')

Medium_Temperature_Hot_Water_Supply_Temperature_Load_Shed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Supply_Temperature_Load_Shed_Setpoint')

Medium_Temperature_Hot_Water_Supply_Temperature_Load_Shed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Supply_Temperature_Load_Shed_Status')

Medium_Temperature_Hot_Water_Supply_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Supply_Temperature_Low_Reset_Setpoint')

Medium_Temperature_Hot_Water_Supply_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Medium_Temperature_Hot_Water_Supply_Temperature_Sensor')

Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Meter')

Methane_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Methane_Level_Sensor')

Min_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Air_Flow_Setpoint_Limit')

Min_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Air_Temperature_Setpoint')

Min_Chilled_Water_Differential_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Chilled_Water_Differential_Pressure_Setpoint_Limit')

Min_Cooling_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Cooling_Discharge_Air_Flow_Setpoint_Limit')

Min_Cooling_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Cooling_Supply_Air_Flow_Setpoint_Limit')

Min_Discharge_Air_Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Discharge_Air_Static_Pressure_Setpoint_Limit')

Min_Discharge_Air_Temperature_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Discharge_Air_Temperature_Setpoint_Limit')

Min_Fresh_Air_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Fresh_Air_Setpoint_Limit')

Min_Heating_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Heating_Discharge_Air_Flow_Setpoint_Limit')

Min_Heating_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Heating_Supply_Air_Flow_Setpoint_Limit')

Min_Hot_Water_Differential_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Hot_Water_Differential_Pressure_Setpoint_Limit')

Min_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Limit')

Min_Occupied_Cooling_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Occupied_Cooling_Discharge_Air_Flow_Setpoint_Limit')

Min_Occupied_Cooling_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Occupied_Cooling_Supply_Air_Flow_Setpoint_Limit')

Min_Occupied_Heating_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Occupied_Heating_Discharge_Air_Flow_Setpoint_Limit')

Min_Occupied_Heating_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Occupied_Heating_Supply_Air_Flow_Setpoint_Limit')

Min_Outside_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Outside_Air_Flow_Setpoint_Limit')

Min_Position_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Position_Setpoint_Limit')

Min_Speed_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Speed_Setpoint_Limit')

Min_Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Static_Pressure_Setpoint_Limit')

Min_Supply_Air_Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Supply_Air_Static_Pressure_Setpoint_Limit')

Min_Temperature_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Temperature_Setpoint_Limit')

Min_Unoccupied_Cooling_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Unoccupied_Cooling_Discharge_Air_Flow_Setpoint_Limit')

Min_Unoccupied_Cooling_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Unoccupied_Cooling_Supply_Air_Flow_Setpoint_Limit')

Min_Unoccupied_Heating_Discharge_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Unoccupied_Heating_Discharge_Air_Flow_Setpoint_Limit')

Min_Unoccupied_Heating_Supply_Air_Flow_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Unoccupied_Heating_Supply_Air_Flow_Setpoint_Limit')

Min_Water_Level_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Water_Level_Alarm')

Min_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Min_Water_Temperature_Setpoint')

Mixed_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air')

Mixed_Air_Filter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air_Filter')

Mixed_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air_Flow_Sensor')

Mixed_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air_Humidity_Sensor')

Mixed_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air_Humidity_Setpoint')

Mixed_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air_Temperature_Sensor')

Mixed_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Air_Temperature_Setpoint')

Mixed_Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mixed_Damper')

Mode_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mode_Command')

Mode_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Mode_Status')

Motion_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motion_Sensor')

Motor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor')

Motor_Control_Center: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor_Control_Center')

Motor_Current_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor_Current_Sensor')

Motor_Direction_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor_Direction_Status')

Motor_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor_On_Off_Status')

Motor_Speed_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor_Speed_Sensor')

Motor_Torque_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Motor_Torque_Sensor')

NO2_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#NO2_Level_Sensor')

NVR: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#NVR')

Natural_Gas: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Natural_Gas')

Natural_Gas_Boiler: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Natural_Gas_Boiler')

Network_Video_Recorder: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Network_Video_Recorder')

No_Water_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#No_Water_Alarm')

Noncondensing_Natural_Gas_Boiler: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Noncondensing_Natural_Gas_Boiler')

Occupancy_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupancy_Command')

Occupancy_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupancy_Sensor')

Occupancy_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupancy_Status')

Occupied_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Air_Temperature_Setpoint')

Occupied_Cooling_Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Cooling_Discharge_Air_Flow_Setpoint')

Occupied_Cooling_Supply_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Cooling_Supply_Air_Flow_Setpoint')

Occupied_Cooling_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Cooling_Temperature_Deadband_Setpoint')

Occupied_Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Discharge_Air_Flow_Setpoint')

Occupied_Discharge_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Discharge_Air_Temperature_Setpoint')

Occupied_Heating_Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Heating_Discharge_Air_Flow_Setpoint')

Occupied_Heating_Supply_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Heating_Supply_Air_Flow_Setpoint')

Occupied_Heating_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Heating_Temperature_Deadband_Setpoint')

Occupied_Mode_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Mode_Status')

Occupied_Return_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Return_Air_Temperature_Setpoint')

Occupied_Room_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Room_Air_Temperature_Setpoint')

Occupied_Supply_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Supply_Air_Flow_Setpoint')

Occupied_Supply_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Supply_Air_Temperature_Setpoint')

Occupied_Zone_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Occupied_Zone_Air_Temperature_Setpoint')

Off_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Off_Command')

Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Off_Status')

Office: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Office')

Office_Kitchen: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Office_Kitchen')

Oil: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Oil')

On_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#On_Command')

On_Off_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#On_Off_Command')

On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#On_Off_Status')

On_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#On_Status')

On_Timer_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#On_Timer_Sensor')

Open_Close_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Open_Close_Status')

Open_Heating_Valve_Outside_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Open_Heating_Valve_Outside_Air_Temperature_Setpoint')

Open_Office: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Open_Office')

Operating_Mode_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Operating_Mode_Status')

Outdoor_Area: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outdoor_Area')

Output_Frequency_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Output_Frequency_Sensor')

Output_Voltage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Output_Voltage_Sensor')

Outside: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside')

Outside_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air')

Outside_Air_CO2_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_CO2_Sensor')

Outside_Air_CO_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_CO_Sensor')

Outside_Air_Dewpoint_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Dewpoint_Sensor')

Outside_Air_Enthalpy_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Enthalpy_Sensor')

Outside_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Flow_Sensor')

Outside_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Flow_Setpoint')

Outside_Air_Grains_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Grains_Sensor')

Outside_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Humidity_Sensor')

Outside_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Humidity_Setpoint')

Outside_Air_Lockout_Temperature_Differential_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Lockout_Temperature_Differential_Parameter')

Outside_Air_Lockout_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Lockout_Temperature_Setpoint')

Outside_Air_Temperature_Enable_Differential_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Temperature_Enable_Differential_Sensor')

Outside_Air_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Temperature_High_Reset_Setpoint')

Outside_Air_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Temperature_Low_Reset_Setpoint')

Outside_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Temperature_Sensor')

Outside_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Temperature_Setpoint')

Outside_Air_Wet_Bulb_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Air_Wet_Bulb_Temperature_Sensor')

Outside_Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Damper')

Outside_Face_Surface_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Face_Surface_Temperature_Sensor')

Outside_Face_Surface_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Face_Surface_Temperature_Setpoint')

Outside_Illuminance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Outside_Illuminance_Sensor')

Overload_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Overload_Alarm')

Overridden_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Overridden_Off_Status')

Overridden_On_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Overridden_On_Status')

Overridden_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Overridden_Status')

Override_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Override_Command')

Ozone_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ozone_Level_Sensor')

PAU: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PAU')

PID_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PID_Parameter')

PIR_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PIR_Sensor')

PM10_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PM10_Level_Sensor')

PM10_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PM10_Sensor')

PM1_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PM1_Level_Sensor')

PM1_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PM1_Sensor')

PVT_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PVT_Panel')

PV_Array: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PV_Array')

PV_Current_Output_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PV_Current_Output_Sensor')

PV_Generation_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PV_Generation_System')

PV_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PV_Panel')

Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Parameter')

Parking_Level: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Parking_Level')

Parking_Space: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Parking_Space')

Parking_Structure: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Parking_Structure')

Particulate_Matter_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Particulate_Matter_Sensor')

Passive_Chilled_Beam: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Passive_Chilled_Beam')

Peak_Power_Demand_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Peak_Power_Demand_Sensor')

Photovoltaic_Array: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Photovoltaic_Array')

Photovoltaic_Current_Output_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Photovoltaic_Current_Output_Sensor')

Piezoelectric_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Piezoelectric_Sensor')

PlugStrip: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#PlugStrip')

Plumbing_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Plumbing_Room')

Point: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Point')

Portfolio: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Portfolio')

Position_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Position_Command')

Position_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Position_Limit')

Position_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Position_Sensor')

Potable_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Potable_Water')

Power_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Power_Alarm')

Power_Loss_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Power_Loss_Alarm')

Power_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Power_Sensor')

Prayer_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Prayer_Room')

Pre_Filter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pre_Filter')

Pre_Filter_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pre_Filter_Status')

Preheat_Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Preheat_Demand_Setpoint')

Preheat_Discharge_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Preheat_Discharge_Air_Temperature_Sensor')

Preheat_Hot_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Preheat_Hot_Water_System')

Preheat_Hot_Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Preheat_Hot_Water_Valve')

Preheat_Supply_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Preheat_Supply_Air_Temperature_Sensor')

Pressure_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pressure_Alarm')

Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pressure_Sensor')

Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pressure_Setpoint')

Pressure_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pressure_Status')

Private_Office: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Private_Office')

Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Proportional_Band_Parameter')

Proportional_Gain_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Proportional_Gain_Parameter')

Pump: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pump')

Pump_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pump_Command')

Pump_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pump_On_Off_Status')

Pump_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pump_Room')

Pump_VFD: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Pump_VFD')

Quantity: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Quantity')

RC_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#RC_Panel')

RTU: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#RTU')

RVAV: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#RVAV')

Radiant_Ceiling_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radiant_Ceiling_Panel')

Radiant_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radiant_Panel')

Radiant_Panel_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radiant_Panel_Temperature_Sensor')

Radiant_Panel_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radiant_Panel_Temperature_Setpoint')

Radiation_Hot_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radiation_Hot_Water_System')

Radiator: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radiator')

Radioactivity_Concentration_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radioactivity_Concentration_Sensor')

Radon_Concentration_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Radon_Concentration_Sensor')

Rain_Duration_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Rain_Duration_Sensor')

Rain_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Rain_Sensor')

Rated_Speed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Rated_Speed_Setpoint')

Reactive_Power_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Reactive_Power_Sensor')

Reception: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Reception')

Region: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Region')

Reheat_Hot_Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Reheat_Hot_Water_System')

Reheat_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Reheat_Valve')

Relative_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Relative_Humidity_Sensor')

Relief_Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Relief_Damper')

Relief_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Relief_Fan')

Remotely_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Remotely_On_Off_Status')

Reset_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Reset_Command')

Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Reset_Setpoint')

Rest_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Rest_Room')

Restroom: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Restroom')

Retail_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Retail_Room')

Return_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air')

Return_Air_CO2_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_CO2_Sensor')

Return_Air_CO2_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_CO2_Setpoint')

Return_Air_CO_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_CO_Sensor')

Return_Air_Dewpoint_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Dewpoint_Sensor')

Return_Air_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Differential_Pressure_Sensor')

Return_Air_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Differential_Pressure_Setpoint')

Return_Air_Enthalpy_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Enthalpy_Sensor')

Return_Air_Filter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Filter')

Return_Air_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Flow_Sensor')

Return_Air_Grains_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Grains_Sensor')

Return_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Humidity_Sensor')

Return_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Humidity_Setpoint')

Return_Air_Plenum: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Plenum')

Return_Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Temperature_Alarm')

Return_Air_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Temperature_High_Reset_Setpoint')

Return_Air_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Temperature_Low_Reset_Setpoint')

Return_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Temperature_Sensor')

Return_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Air_Temperature_Setpoint')

Return_Chilled_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Chilled_Water_Temperature_Setpoint')

Return_Condenser_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Condenser_Water')

Return_Condenser_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Condenser_Water_Flow_Sensor')

Return_Condenser_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Condenser_Water_Temperature_Sensor')

Return_Condenser_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Condenser_Water_Temperature_Setpoint')

Return_Damper: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Damper')

Return_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Fan')

Return_Heating_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Heating_Valve')

Return_Hot_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Hot_Water')

Return_Hot_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Hot_Water_Temperature_Setpoint')

Return_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Water')

Return_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Water_Flow_Sensor')

Return_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Water_Temperature_Sensor')

Return_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Return_Water_Temperature_Setpoint')

Riser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Riser')

Rooftop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Rooftop')

Rooftop_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Rooftop_Unit')

Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Room')

Room_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Room_Air_Temperature_Setpoint')

Run_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Run_Enable_Command')

Run_Request_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Run_Request_Status')

Run_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Run_Status')

Run_Time_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Run_Time_Sensor')

Safety_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Safety_Equipment')

Safety_Shower: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Safety_Shower')

Safety_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Safety_System')

Sash_Position_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Sash_Position_Sensor')

Schedule_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Schedule_Temperature_Setpoint')

Security_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Security_Equipment')

Security_Service_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Security_Service_Room')

Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Sensor')

Server_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Server_Room')

Service_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Service_Room')

Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Setpoint')

Shading_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Shading_System')

Shared_Office: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Shared_Office')

Short_Cycle_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Short_Cycle_Alarm')

Shower: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Shower')

Site: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Site')

Smoke_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Smoke_Alarm')

Smoke_Detection_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Smoke_Detection_Alarm')

Solar_Azimuth_Angle_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Solar_Azimuth_Angle_Sensor')

Solar_Radiance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Solar_Radiance_Sensor')

Solar_Thermal_Collector: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Solar_Thermal_Collector')

Solar_Zenith_Angle_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Solar_Zenith_Angle_Sensor')

Solid: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Solid')

Space: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Space')

Space_Heater: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Space_Heater')

Speed_Reset_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Speed_Reset_Command')

Speed_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Speed_Sensor')

Speed_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Speed_Setpoint')

Speed_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Speed_Setpoint_Limit')

Speed_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Speed_Status')

Sports_Service_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Sports_Service_Room')

Stage_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Stage_Enable_Command')

Stage_Riser: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Stage_Riser')

Stages_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Stages_Status')

Staircase: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Staircase')

Standby_CRAC: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Standby_CRAC')

Standby_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Standby_Fan')

Standby_Glycool_Unit_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Standby_Glycool_Unit_On_Off_Status')

Standby_Load_Shed_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Standby_Load_Shed_Command')

Standby_Unit_On_Off_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Standby_Unit_On_Off_Status')

Start_Stop_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Start_Stop_Command')

Start_Stop_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Start_Stop_Status')

Static_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Deadband_Setpoint')

Static_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Integral_Time_Parameter')

Static_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Proportional_Band_Parameter')

Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Sensor')

Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Setpoint')

Static_Pressure_Setpoint_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Setpoint_Limit')

Static_Pressure_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Static_Pressure_Step_Parameter')

Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Status')

Steam: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam')

Steam_Baseboard_Radiator: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_Baseboard_Radiator')

Steam_Distribution: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_Distribution')

Steam_On_Off_Command: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_On_Off_Command')

Steam_Radiator: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_Radiator')

Steam_System: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_System')

Steam_Usage_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_Usage_Sensor')

Steam_Valve: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Steam_Valve')

Step_Parameter: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Step_Parameter')

Storage_Room: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Storage_Room')

Storey: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Storey')

Studio: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Studio')

Substance: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Substance')

Supply_Air: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air')

Supply_Air_Differential_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://
brickschema.org/schema/Brick#Supply_Air_Differential_Pressure_Sensor')

Supply_Air_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://
brickschema.org/schema/Brick#Supply_Air_Differential_Pressure_Setpoint')

Supply_Air_Duct_Pressure_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.
org/schema/Brick#Supply_Air_Duct_Pressure_Status')

Supply_Air_Flow_Demand_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.
org/schema/Brick#Supply_Air_Flow_Demand_Setpoint')

Supply_Air_Flow_Sensor: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Flow_Sensor')

Supply_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Flow_Setpoint')

Supply_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Humidity_Sensor')

Supply_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Humidity_Setpoint')

Supply_Air_Integral_Gain_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Integral_Gain_Parameter')

Supply_Air_Plenum: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Plenum')

Supply_Air_Proportional_Gain_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Proportional_Gain_Parameter')

Supply_Air_Static_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Static_Pressure_Deadband_Setpoint')

Supply_Air_Static_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Static_Pressure_Integral_Time_Parameter')

Supply_Air_Static_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Static_Pressure_Proportional_Band_Parameter')

Supply_Air_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Static_Pressure_Sensor')

Supply_Air_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Static_Pressure_Setpoint')

Supply_Air_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Alarm')

Supply_Air_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Deadband_Setpoint')

Supply_Air_Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_High_Reset_Setpoint')

Supply_Air_Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Low_Reset_Setpoint')

Supply_Air_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Proportional_Band_Parameter')

Supply_Air_Temperature_Reset_Differential_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Reset_Differential_Setpoint')

Supply_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Sensor')

Supply_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Setpoint')

Supply_Air_Temperature_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Temperature_Step_Parameter')

Supply_Air_Velocity_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Air_Velocity_Pressure_Sensor')

Supply_Chilled_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Chilled_Water')

Supply_Chilled_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Chilled_Water_Temperature_Setpoint')

Supply_Condenser_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Condenser_Water')

Supply_Condenser_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Condenser_Water_Flow_Sensor')

Supply_Condenser_Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Condenser_Water_Temperature_Sensor')

Supply_Condenser_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Condenser_Water_Temperature_Setpoint')

Supply_Fan: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Fan')

Supply_Hot_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Hot_Water')

Supply_Hot_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Hot_Water_Temperature_Setpoint')

Supply_Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water')

Supply_Water_Differential_Pressure_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Differential_Pressure_Deadband_Setpoint')

Supply_Water_Differential_Pressure_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Differential_Pressure_Integral_Time_Parameter')

Supply_Water_Differential_Pressure_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Differential_Pressure_Proportional_Band_Parameter')

Supply_Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Flow_Sensor')

Supply_Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Flow_Setpoint')

Supply_Water_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Temperature_Alarm')

Supply_Water_Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Temperature_Deadband_Setpoint')

Supply_Water_Temperature_Integral_Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Temperature_Integral_Time_Parameter')

Supply_Water_Temperature_Proportional_Band_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Temperature_Proportional_Band_Parameter')

Supply_Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Supply_Water_Temperature_Setpoint')

Surveillance_Camera: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Surveillance_Camera')

Switch: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Switch')

Switch_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Switch_Room')

Switchgear: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Switchgear')

System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#System')

System_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#System_Enable_Command')

System_Shutdown_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#System_Shutdown_Status')

System_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#System_Status')

TABS_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#TABS_Panel')

TETRA_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#TETRA_Room')

TVOC_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#TVOC_Level_Sensor')

TVOC_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#TVOC_Sensor')

Team_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Team_Room')

Telecom_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Telecom_Room')

Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Alarm')

Temperature_Deadband_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Deadband_Setpoint')

Temperature_Differential_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Differential_Reset_Setpoint')

Temperature_High_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_High_Reset_Setpoint')

Temperature_Low_Reset_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Low_Reset_Setpoint')

Temperature_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Parameter')

Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Sensor')

Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Setpoint')

Temperature_Step_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Step_Parameter')

Temperature_Tolerance_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temperature_Tolerance_Parameter')

Temporary_Occupancy_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Temporary_Occupancy_Status')

Terminal_Unit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Terminal_Unit')

Thermal_Power_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Thermal_Power_Meter')

Thermal_Power_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Thermal_Power_Sensor')

Thermally_Activated_Building_System_Panel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Thermally_Activated_Building_System_Panel')

Thermostat: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Thermostat')

Ticketing_Booth: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ticketing_Booth')

Time_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Time_Parameter')

Time_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Time_Setpoint')

Tolerance_Parameter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Tolerance_Parameter')

Torque_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Torque_Sensor')

Touchpanel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Touchpanel')

Trace_Heat_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Trace_Heat_Sensor')

Transformer: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Transformer')

Transformer_Room: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Transformer_Room')

Tunnel: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Tunnel')

Underfloor_Air_Plenum: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Underfloor_Air_Plenum')

Underfloor_Air_Plenum_Static_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Underfloor_Air_Plenum_Static_Pressure_Sensor')

Underfloor_Air_Plenum_Static_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Underfloor_Air_Plenum_Static_Pressure_Setpoint')

Underfloor_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Underfloor_Air_Temperature_Sensor')

Unit_Failure_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unit_Failure_Alarm')

Unoccupied_Air_Temperature_Cooling_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Air_Temperature_Cooling_Setpoint')

Unoccupied_Air_Temperature_Heating_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Air_Temperature_Heating_Setpoint')

Unoccupied_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Air_Temperature_Setpoint')

Unoccupied_Cooling_Discharge_Air_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Cooling_Discharge_Air_Flow_Setpoint')

Unoccupied_Discharge_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Discharge_Air_Temperature_Setpoint')

Unoccupied_Load_Shed_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Load_Shed_Command')

Unoccupied_Return_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Return_Air_Temperature_Setpoint')

Unoccupied_Room_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Room_Air_Temperature_Setpoint')

Unoccupied_Supply_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Supply_Air_Temperature_Setpoint')

Unoccupied_Zone_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Unoccupied_Zone_Air_Temperature_Setpoint')

Usage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Usage_Sensor')

VAV: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#VAV')

VFD: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#VFD')

VFD_Enable_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#VFD_Enable_Command')

Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Valve')

Valve_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Valve_Command')

Valve_Position_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Valve_Position_Sensor')

Variable_Air_Volume_Box: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Variable_Air_Volume_Box')

Variable_Air_Volume_Box_With_Reheat: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Variable_Air_Volume_Box_With_Reheat')

Variable_Frequency_Drive: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Variable_Frequency_Drive')

Velocity_Pressure_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Velocity_Pressure_Sensor')

Velocity_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Velocity_Pressure_Setpoint')

Vent_Operating_Mode_Status: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Vent_Operating_Mode_Status')

Ventilation_Air_Flow_Ratio_Limit: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ventilation_Air_Flow_Ratio_Limit')

Ventilation_Air_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Ventilation_Air_System')

Vertical_Space: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Vertical_Space')

Video_Intercom: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Video_Intercom')

Video_Surveillance_Equipment: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Video_Surveillance_Equipment')

Visitor_Lobby: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Visitor_Lobby')

Voltage_Imbalance_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Voltage_Imbalance_Sensor')

Voltage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Voltage_Sensor')

Wardrobe: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Wardrobe')

Warm_Cool_Adjust_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Warm_Cool_Adjust_Sensor')

Warmest_Zone_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Warmest_Zone_Air_Temperature_Sensor')

Waste_Storage: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Waste_Storage')

Water: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water')

Water_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Alarm')

Water_Differential_Pressure_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Differential_Pressure_Setpoint')

Water_Differential_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Differential_Temperature_Sensor')

Water_Differential_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Differential_Temperature_Setpoint')

Water_Distribution: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Distribution')

Water_Flow_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Flow_Sensor')

Water_Flow_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Flow_Setpoint')

Water_Heater: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Heater')

Water_Level_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Level_Alarm')

Water_Level_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Level_Sensor')

Water_Loop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Loop')

Water_Loss_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Loss_Alarm')

Water_Meter: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Meter')

Water_Pump: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Pump')

Water_System: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_System')

Water_Tank: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Tank')

Water_Temperature_Alarm: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Temperature_Alarm')

Water_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Temperature_Sensor')

Water_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Temperature_Setpoint')

Water_Usage_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Usage_Sensor')

Water_Valve: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Water_Valve')

Weather_Station: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Weather_Station')

Wind_Direction_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Wind_Direction_Sensor')

Wind_Speed_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Wind_Speed_Sensor')

Wing: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Wing')

Workshop: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Workshop')

Zone: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone')

Zone_Air: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air')

Zone_Air_Cooling_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Cooling_Temperature_Setpoint')

Zone_Air_Dewpoint_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Dewpoint_Sensor')

Zone_Air_Heating_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Heating_Temperature_Setpoint')

Zone_Air_Humidity_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Humidity_Sensor')

Zone_Air_Humidity_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Humidity_Setpoint')

Zone_Air_Temperature_Sensor: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Temperature_Sensor')

Zone_Air_Temperature_Setpoint: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Air_Temperature_Setpoint')

Zone_Standby_Load_Shed_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Standby_Load_Shed_Command')

Zone_Unoccupied_Load_Shed_Command: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#Zone_Unoccupied_Load_Shed_Command')

aggregate: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#aggregate')

area: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#area')

azimuth: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#azimuth')

buildingPrimaryFunction: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#buildingPrimaryFunction')

buildingThermalTransmittance: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#buildingThermalTransmittance')

conversionEfficiency: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#conversionEfficiency')

coolingCapacity: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#coolingCapacity')

coordinates: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#coordinates')

currentFlowType: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#currentFlowType')

electricalPhaseCount: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#electricalPhaseCount')

electricalPhases: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#electricalPhases')

feeds: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#feeds')

feedsAir: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#feedsAir')

grossArea: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#grossArea')

hasAddress: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasAddress')

hasAssociatedTag: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasAssociatedTag')

hasInputSubstance: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasInputSubstance')

hasLocation: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasLocation')

hasOutputSubstance: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasOutputSubstance')

hasPart: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasPart')

hasPoint: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasPoint')

hasQUDTReference: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasQUDTReference')

hasTag: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasTag')

hasTimeseriesId: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasTimeseriesId')

hasUnit: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#hasUnit')

isAssociatedWith: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#isAssociatedWith')

isFedBy: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#isFedBy')

isLocationOf: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#isLocationOf')

isMeasuredBy: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#isMeasuredBy')

isPartOf: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#isPartOf')

isPointOf: *URIRef* =
rdflib.term.URIRef('https://brickschema.org/schema/Brick#isPointOf')

isRegulatedBy: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#isRegulatedBy')

isTagOf: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#isTagOf')

latitude: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#latitude')

longitude: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#longitude')

measuredModuleConversionEfficiency: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#measuredModuleConversionEfficiency')

measuredPowerOutput: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#measuredPowerOutput')

measures: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#measures')

netArea: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#netArea')

operationalStage: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#operationalStage')

operationalStageCount: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#operationalStageCount')

panelArea: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#panelArea')

powerComplexity: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#powerComplexity')

powerFlow: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#powerFlow')

ratedModuleConversionEfficiency: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#ratedModuleConversionEfficiency')

ratedPowerOutput: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#ratedPowerOutput')

regulates: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#regulates')

storedAt: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#storedAt')

temperatureCoefficientofPmax: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#temperatureCoefficientofPmax')

thermalTransmittance: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#thermalTransmittance')

tilt: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#tilt')

timeseries: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#timeseries')

value: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#value')

volume: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#volume')

yearBuilt: *URIRef* = rdflib.term.URIRef('https://brickschema.org/schema/Brick#yearBuilt')

class rdflib.**CSVW**

Bases: *DefinedNamespace*

CSVW Namespace Vocabulary Terms

This document describes the RDFS vocabulary description used in the Metadata Vocabulary for Tabular Data [[tabular-metadata]] along with the default JSON-LD Context.

Generated from: http://www.w3.org/ns/csvw Date: 2020-05-26 14:19:58.184766

Cell: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Cell')

Column: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Column')

Datatype: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Datatype')

Dialect: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Dialect')

Direction: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Direction')

ForeignKey: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#ForeignKey')

JSON: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#JSON')

NumericFormat: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#NumericFormat')

Row: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Row')

Schema: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Schema')

Table: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Table')

TableGroup: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#TableGroup')

TableReference: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#TableReference')

Transformation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#Transformation')

aboutUrl: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#aboutUrl')

auto: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#auto')

base: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#base')

column: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#column')

columnReference: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#columnReference')

commentPrefix: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#commentPrefix')

csvEncodedTabularData: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#csvEncodedTabularData')

datatype: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#datatype')

decimalChar: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#decimalChar')

default: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#default')

delimiter: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#delimiter')

describes: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#describes')

dialect: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#dialect')

doubleQuote: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#doubleQuote')

encoding: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#encoding')

foreignKey: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#foreignKey')

format: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#format')

groupChar: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#groupChar')

header: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#header')

headerRowCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#headerRowCount')

inherit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#inherit')

lang: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#lang')

length: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#length')

lineTerminators: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#lineTerminators')

ltr: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#ltr')

maxExclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#maxExclusive')

maxInclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#maxInclusive')

maxLength: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#maxLength')

minExclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#minExclusive')

minInclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#minInclusive')

minLength: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#minLength')

name: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#name')

note: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#note')

null: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#null')

ordered: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#ordered')

pattern: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#pattern')

primaryKey: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#primaryKey')

propertyUrl: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#propertyUrl')

quoteChar: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#quoteChar')

reference: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#reference')

referencedRow: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/csvw#referencedRow')

required: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#required')

resource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#resource')

row: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#row')

rowTitle: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#rowTitle')

rownum: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#rownum')

rtl: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#rtl')

schemaReference: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/csvw#schemaReference')

scriptFormat: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#scriptFormat')

separator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#separator')

skipBlankRows: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/csvw#skipBlankRows')

skipColumns: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#skipColumns')

skipInitialSpace: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/csvw#skipInitialSpace')

skipRows: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#skipRows')

source: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#source')

suppressOutput: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/csvw#suppressOutput')

table: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#table')

tableDirection: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/csvw#tableDirection')

tableSchema: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#tableSchema')

tabularMetadata: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#tabularMetadata')

targetFormat: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#targetFormat')

textDirection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#textDirection')

title: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#title')

transformations: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#transformations')

trim: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#trim')

uriTemplate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#uriTemplate')

url: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#url')

valueUrl: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#valueUrl')

virtual: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/csvw#virtual')

class rdflib.**ConjunctiveGraph**(*store='default'*, *identifier=None*, *default_graph_base=None*)

Bases: *Graph*

A ConjunctiveGraph is an (unnamed) aggregation of all the named graphs in a store.

It has a `default` graph, whose name is associated with the graph throughout its life. *__init__()* can take an identifier to use as the name of this default graph or it will assign a BNode.

All methods that add triples work against this default graph.

All queries are carried out against the union of all graphs.

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **default_graph_base** (Optional[str]) –

__annotations__ = {'__identifier': '_ContextIdentifierType', '__store': 'Store', 'default_context': '_ContextType'}

__contains__(*triple_or_quad*)

Support for 'triple/quad in graph' syntax

Parameters
triple_or_quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]]]) –

Return type
bool

__init__(*store='default'*, *identifier=None*, *default_graph_base=None*)

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –

- **default_graph_base** (Optional[str]) –

__len__()

Number of triples in the entire conjunctive graph

Return type
int

__module__ = 'rdflib.graph'

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Graph*], Tuple[*Store*, *IdentifiedNode*]]

__str__()

Return str(self).

Return type
str

add(*triple_or_quad*)

Add a triple or quad to the store.

if a triple is given it is added to the default context

Parameters

- **self** (TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)) –
- **triple_or_quad** (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]]]) –

Return type
TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)

addN(*quads*)

Add a sequence of triples with context

Parameters

- **self** (TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)) –
- **quads** (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)

context_id(*uri*, *context_id=None*)

URI#context

Parameters

- **uri** (str) –
- **context_id** (Optional[str]) –

Return type
URIRef

contexts(*triple=None*)

Iterate over all contexts in the graph

If triple is specified, iterate over all contexts the triple is in.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*Graph*, None, None]

default_context: *Graph*

get_context(*identifier*, *quoted=False*, *base=None*)

Return a context graph for the given identifier

identifier must be a URIRef or BNode.

Parameters

- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **quoted** (bool) –
- **base** (Optional[str]) –

Return type
Graph

get_graph(*identifier*)

Returns the graph identified by given identifier

Parameters
identifier (*IdentifiedNode*) –

Return type
Optional[*Graph*]

parse(*source=None*, *publicID=None*, *format=None*, *location=None*, *file=None*, *data=None*, ***args*)

Parse source adding the resulting triples to its own context (sub graph of this graph).

See *rdflib.graph.Graph.parse()* for documentation on arguments.

Returns

The graph into which the source was parsed. In the case of n3 it returns the root context.

> **Caution:** This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.
>
> When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –

- **location** (Optional[str]) –
- **file** (Union[BinaryIO, TextIO, None]) –
- **data** (Union[str, bytes, None]) –
- **args** (Any) –

Return type
Graph

quads(*triple_or_quad=None*)

Iterate over all the quads in the entire conjunctive graph

Parameters
triple_or_quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]], None]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]], None, None]

remove(*triple_or_quad*)

Removes a triple or quads

if a triple is given it is removed from all contexts

a quad is removed from the given context only

Parameters

- **self** (TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)) –
- **triple_or_quad** (Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Node*, *Node*, Optional[*Graph*]]]) –

Return type
TypeVar(_ConjunctiveGraphT, bound= *ConjunctiveGraph*)

remove_context(*context*)

Removes the given context from the graph

Parameters
context (*Graph*) –

Return type
None

triples(*triple_or_quad: Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[Graph]]], context: Optional[Graph] = None*) → Generator[Tuple[*Node*, *Node*, *Node*], None, None]

triples(*triple_or_quad: Union[Tuple[Optional[*Node*], Path, Optional[*Node*]], Tuple[Optional[*Node*], Path, Optional[*Node*], Optional[Graph]]], context: Optional[Graph] = None*) → Generator[Tuple[*Node*, *Path*, *Node*], None, None]

triples(*triple_or_quad: Union[Tuple[Optional[*Node*], Optional[Union[Path, *Node*]], Optional[*Node*]], Tuple[Optional[*Node*], Optional[Union[Path, *Node*]], Optional[*Node*], Optional[Graph]]], context: Optional[Graph] = None*) → Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

Iterate over all the triples in the entire conjunctive graph

For legacy reasons, this can take the context to query either as a fourth element of the quad, or as the explicit context keyword parameter. The kw param takes precedence.

Parameters

- **triple_or_quad** (Union[Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]], Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*], Optional[*Graph*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

triples_choices(*triple*, *context=None*)

Iterate over all the triples in the entire conjunctive graph

Parameters

- **triple** (Union[Tuple[List[*Node*], *Node*, *Node*], Tuple[*Node*, List[*Node*], *Node*], Tuple[*Node*, *Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

class rdflib.**DC**

Bases: *DefinedNamespace*

Dublin Core Metadata Element Set, Version 1.1

Generated from: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/dublin_core_elements.ttl
Date: 2020-05-26 14:19:58.671906

contributor: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/contributor')

coverage: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/coverage')

creator: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/creator')

date: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/date')

description: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/description')

format: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/format')

identifier: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/identifier')

language: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/language')

publisher: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/publisher')

relation: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/relation')

rights: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/rights')

source: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/source')

subject: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/subject')

title: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/title')

type: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/elements/1.1/type')

class rdflib.**DCAT**

Bases: *DefinedNamespace*

The data catalog vocabulary

DCAT is an RDF vocabulary designed to facilitate interoperability between data catalogs published on the Web. By using DCAT to describe datasets in data catalogs, publishers increase discoverability and enable applications easily to consume metadata from multiple catalogs. It further enables decentralized publishing of catalogs and facilitates federated dataset search across sites. Aggregated DCAT metadata can serve as a manifest file to facilitate digital preservation. DCAT is defined at http://www.w3.org/TR/vocab-dcat/. Any variance between that normative document and this schema is an error in this schema.

Generated from: https://www.w3.org/ns/dcat2.ttl Date: 2020-05-26 14:19:59.985854

Catalog: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#Catalog')

CatalogRecord: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#CatalogRecord')

DataService: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#DataService')

Dataset: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#Dataset')

Distribution: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#Distribution')

Relationship: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#Relationship')

Resource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#Resource')

Role: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#Role')

accessService: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#accessService')

accessURL: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#accessURL')

bbox: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#bbox')

byteSize: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#byteSize')

catalog: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#catalog')

centroid: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#centroid')

compressFormat: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#compressFormat')

contactPoint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#contactPoint')

dataset: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#dataset')

distribution: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#distribution')

downloadURL: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#downloadURL')

endDate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#endDate')

endpointDescription: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#endpointDescription')

endpointURL: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#endpointURL')

hadRole: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#hadRole')

keyword: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#keyword')

landingPage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#landingPage')

mediaType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#mediaType')

packageFormat: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#packageFormat')

qualifiedRelation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#qualifiedRelation')

record: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#record')

servesDataset: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#servesDataset')

service: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#service')

spatialResolutionInMeters: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#spatialResolutionInMeters')

startDate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#startDate')

temporalResolution: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#temporalResolution')

theme: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#theme')

themeTaxonomy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/dcat#themeTaxonomy')

class rdflib.**DCMITYPE**

Bases: *DefinedNamespace*

DCMI Type Vocabulary

Generated from: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/dublin_core_type.ttl Date: 2020-05-26 14:19:59.084150

Collection: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Collection')

Dataset: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Dataset')

Event: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Event')

Image: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Image')

InteractiveResource: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/InteractiveResource')

MovingImage: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/MovingImage')

PhysicalObject: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/PhysicalObject')

Service: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Service')

Software: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Software')

Sound: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Sound')

StillImage: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/StillImage')

Text: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/dcmitype/Text')

class rdflib.DCTERMS

Bases: *DefinedNamespace*

DCMI Metadata Terms - other

Generated from: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/dublin_core_terms.ttl
Date: 2020-05-26 14:20:00.590514

Agent: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Agent')

AgentClass: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/AgentClass')

BibliographicResource: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/BibliographicResource')

Box: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Box')

DCMIType: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/DCMIType')

DDC: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/DDC')

FileFormat: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/FileFormat')

Frequency: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Frequency')

IMT: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/IMT')

ISO3166: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/ISO3166')

Jurisdiction: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Jurisdiction')

LCC: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/LCC')

LCSH: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/LCSH')

LicenseDocument: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/LicenseDocument')

LinguisticSystem: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/LinguisticSystem')

Location: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Location')

LocationPeriodOrJurisdiction: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/LocationPeriodOrJurisdiction')

MESH: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/MESH')

MediaType: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/MediaType')

MediaTypeOrExtent: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/MediaTypeOrExtent')

MethodOfAccrual: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/MethodOfAccrual')

MethodOfInstruction: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/MethodOfInstruction')

NLM: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/NLM')

Period: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Period')

PeriodOfTime: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/PeriodOfTime')

PhysicalMedium: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/PhysicalMedium')

PhysicalResource: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/PhysicalResource')

Point: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Point')

Policy: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Policy')

ProvenanceStatement: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/ProvenanceStatement')

RFC1766: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/RFC1766')

RFC3066: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/RFC3066')

RFC4646: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/RFC4646')

RFC5646: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/RFC5646')

RightsStatement: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/RightsStatement')

SizeOrDuration: *URIRef* =
rdflib.term.URIRef('http://purl.org/dc/terms/SizeOrDuration')

Standard: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/Standard')

TGN: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/TGN')

UDC: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/UDC')

URI: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/URI')

W3CDTF: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/W3CDTF')

abstract: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/abstract')

accessRights: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/accessRights')

accrualMethod: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/accrualMethod')

accrualPeriodicity: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/accrualPeriodicity')

accrualPolicy: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/accrualPolicy')

alternative: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/alternative')

audience: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/audience')

available: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/available')

bibliographicCitation: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/bibliographicCitation')

conformsTo: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/conformsTo')

contributor: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/contributor')

coverage: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/coverage')

created: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/created')

creator: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/creator')

date: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/date')

dateAccepted: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/dateAccepted')

dateCopyrighted: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/dateCopyrighted')

dateSubmitted: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/dateSubmitted')

description: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/description')

educationLevel: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/educationLevel')

extent: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/extent')

format: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/format')

hasFormat: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/hasFormat')

hasPart: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/hasPart')

hasVersion: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/hasVersion')

identifier: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/identifier')

instructionalMethod: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/instructionalMethod')

isFormatOf: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/isFormatOf')

isPartOf: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/isPartOf')

isReferencedBy: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/isReferencedBy')

isReplacedBy: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/isReplacedBy')

isRequiredBy: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/isRequiredBy')

isVersionOf: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/isVersionOf')

issued: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/issued')

language: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/language')

license: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/license')

mediator: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/mediator')

medium: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/medium')

modified: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/modified')

provenance: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/provenance')

publisher: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/publisher')

references: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/references')

relation: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/relation')

replaces: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/replaces')

requires: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/requires')

rights: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/rights')

rightsHolder: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/rightsHolder')

source: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/source')

spatial: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/spatial')

subject: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/subject')

tableOfContents: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/tableOfContents')

temporal: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/temporal')

title: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/title')

type: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/type')

valid: *URIRef* = rdflib.term.URIRef('http://purl.org/dc/terms/valid')

class rdflib.DOAP

Bases: *DefinedNamespace*

Description of a Project (DOAP) vocabulary

The Description of a Project (DOAP) vocabulary, described using W3C RDF Schema and the Web Ontology Language.

Generated from: http://usefulinc.com/ns/doap Date: 2020-05-26 14:20:01.307972

ArchRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#ArchRepository')

BKRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#BKRepository')

BazaarBranch: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#BazaarBranch')

CVSRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#CVSRepository')

DarcsRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#DarcsRepository')

GitBranch: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#GitBranch')

GitRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#GitRepository')

HgRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#HgRepository')

Project: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#Project')

Repository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#Repository')

SVNRepository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#SVNRepository')

Specification: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#Specification')

Version: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#Version')

audience: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#audience')

blog: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#blog')

browse: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#browse')

category: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#category')

created: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#created')

description: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#description')

developer: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#developer')

documenter: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#documenter')

helper: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#helper')

homepage: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#homepage')

implements: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#implements')

language: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#language')

license: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#license')

location: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#location')

maintainer: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#maintainer')

module: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#module')

name: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#name')

os: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#os')

platform: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#platform')

release: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#release')

repository: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#repository')

repositoryOf: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#repositoryOf')

revision: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#revision')

screenshots: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#screenshots')

shortdesc: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#shortdesc')

tester: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#tester')

translator: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#translator')

vendor: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#vendor')

wiki: *URIRef* = rdflib.term.URIRef('http://usefulinc.com/ns/doap#wiki')

class rdflib.**Dataset**(*store='default'*, *default_union=False*, *default_graph_base=None*)

Bases: *ConjunctiveGraph*

RDF 1.1 Dataset. Small extension to the Conjunctive Graph: - the primary term is graphs in the datasets and not contexts with quads, so there is a separate method to set/retrieve a graph in a dataset and operate with graphs - graphs cannot be identified with blank nodes - added a method to directly add a single quad

Examples of usage:

```
>>> # Create a new Dataset
>>> ds = Dataset()
>>> # simple triples goes to default graph
>>> ds.add((URIRef("http://example.org/a"),
...    URIRef("http://www.example.org/b"),
```

(continues on next page)

(continued from previous page)

```
...     Literal("foo")))
<Graph identifier=... (<class 'rdflib.graph.Dataset'>)>
>>>
>>> # Create a graph in the dataset, if the graph name has already been
>>> # used, the corresponding graph will be returned
>>> # (ie, the Dataset keeps track of the constituent graphs)
>>> g = ds.graph(URIRef("http://www.example.com/gr"))
>>>
>>> # add triples to the new graph as usual
>>> g.add(
...     (URIRef("http://example.org/x"),
...     URIRef("http://example.org/y"),
...     Literal("bar")) )
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> # alternatively: add a quad to the dataset -> goes to the graph
>>> ds.add(
...     (URIRef("http://example.org/x"),
...     URIRef("http://example.org/z"),
...     Literal("foo-bar"),g) )
<Graph identifier=... (<class 'rdflib.graph.Dataset'>)>
>>>
>>> # querying triples return them all regardless of the graph
>>> for t in ds.triples((None,None,None)):
...     print(t)
(rdflib.term.URIRef("http://example.org/a"),
 rdflib.term.URIRef("http://www.example.org/b"),
 rdflib.term.Literal("foo"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"))
>>>
>>> # querying quads() return quads; the fourth argument can be unrestricted
>>> # (None) or restricted to a graph
>>> for q in ds.quads((None, None, None, None)):
...     print(q)
(rdflib.term.URIRef("http://example.org/a"),
 rdflib.term.URIRef("http://www.example.org/b"),
 rdflib.term.Literal("foo"),
 None)
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
>>>
>>> # unrestricted looping is equivalent to iterating over the entire Dataset
```

(continues on next page)

(continued from previous page)

```
>>> for q in ds:
...     print(q)
(rdflib.term.URIRef("http://example.org/a"),
 rdflib.term.URIRef("http://www.example.org/b"),
 rdflib.term.Literal("foo"),
 None)
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
>>>
>>> # resticting iteration to a graph:
>>> for q in ds.quads((None, None, None, g)):
...     print(q)
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/y"),
 rdflib.term.Literal("bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
(rdflib.term.URIRef("http://example.org/x"),
 rdflib.term.URIRef("http://example.org/z"),
 rdflib.term.Literal("foo-bar"),
 rdflib.term.URIRef("http://www.example.com/gr"))
>>> # Note that in the call above -
>>> # ds.quads((None,None,None,"http://www.example.com/gr"))
>>> # would have been accepted, too
>>>
>>> # graph names in the dataset can be queried:
>>> for c in ds.graphs():
...     print(c)  # doctest:
DEFAULT
http://www.example.com/gr
>>> # A graph can be created without specifying a name; a skolemized genid
>>> # is created on the fly
>>> h = ds.graph()
>>> for c in ds.graphs():
...     print(c)
DEFAULT
https://rdflib.github.io/.well-known/genid/rdflib/N...
http://www.example.com/gr
>>> # Note that the Dataset.graphs() call returns names of empty graphs,
>>> # too. This can be restricted:
>>> for c in ds.graphs(empty=False):
...     print(c)
DEFAULT
http://www.example.com/gr
>>>
>>> # a graph can also be removed from a dataset via ds.remove_graph(g)
```

New in version 4.0.

Parameters

- **store** (Union[*Store*, str]) –
- **default_union** (bool) –
- **default_graph_base** (Optional[str]) –

__annotations__ = {'__identifier': '_ContextIdentifierType', '__store': 'Store', 'default_context': '_ContextType'}

__getstate__()

Return type
Tuple[*Store*, *IdentifiedNode*, *Graph*, bool]

__init__(*store='default'*, *default_union=False*, *default_graph_base=None*)

Parameters

- **store** (Union[*Store*, str]) –
- **default_union** (bool) –
- **default_graph_base** (Optional[str]) –

__iter__()

Iterates over all quads in the store

Return type
Generator[Tuple[*Node*, *Node*, *Node*, Optional[*IdentifiedNode*]], None, None]

__module__ = 'rdflib.graph'

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Dataset*], Tuple[*Store*, bool]]

__setstate__(*state*)

Parameters
state (Tuple[*Store*, *IdentifiedNode*, *Graph*, bool]) –

Return type
None

__str__()

Return str(self).

Return type
str

add_graph(*g*)

alias of graph for consistency

Parameters
g (Union[*IdentifiedNode*, *Graph*, str, None]) –

Return type
Graph

contexts(*triple=None*)

Iterate over all contexts in the graph

If triple is specified, iterate over all contexts the triple is in.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*Graph*, None, None]

default_context: *Graph*

graph(*identifier=None*, *base=None*)

Parameters

- **identifier** (Union[*IdentifiedNode*, *Graph*, str, None]) –
- **base** (Optional[str]) –

Return type
Graph

graphs(*triple=None*)

Iterate over all contexts in the graph

If triple is specified, iterate over all contexts the triple is in.

Parameters
triple (Optional[Tuple[*Node*, *Node*, *Node*]]) –

Return type
Generator[*Graph*, None, None]

parse(*source=None*, *publicID=None*, *format=None*, *location=None*, *file=None*, *data=None*, ***args*)

> **Caution:** This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.
>
> When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –
- **location** (Optional[str]) –
- **file** (Union[BinaryIO, TextIO, None]) –
- **data** (Union[str, bytes, None]) –
- **args** (Any) –

Return type
Graph

quads(*quad=None*)

Iterate over all the quads in the entire conjunctive graph

Parameters

quad (Union[Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]], Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*], Optional[*Graph*]], None]) –

Return type

Generator[Tuple[*Node*, *Node*, *Node*, Optional[*IdentifiedNode*]], None, None]

remove_graph(*g*)

Parameters

- **self** (TypeVar(_DatasetT, bound= *Dataset*)) –
- **g** (Union[*IdentifiedNode*, *Graph*, str, None]) –

Return type

TypeVar(_DatasetT, bound= *Dataset*)

class rdflib.**FOAF**

Bases: *DefinedNamespace*

Friend of a Friend (FOAF) vocabulary

The Friend of a Friend (FOAF) RDF vocabulary, described using W3C RDF Schema and the Web Ontology Language.

Generated from: http://xmlns.com/foaf/spec/index.rdf Date: 2020-05-26 14:20:01.597998

Agent: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Agent')

Document: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Document')

Group: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Group')

Image: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Image')

LabelProperty: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/LabelProperty')

OnlineAccount: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/OnlineAccount')

OnlineChatAccount: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/OnlineChatAccount')

OnlineEcommerceAccount: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/OnlineEcommerceAccount')

OnlineGamingAccount: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/OnlineGamingAccount')

Organization: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Organization')

Person: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Person')

PersonalProfileDocument: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/PersonalProfileDocument')

Project: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/Project')

account: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/account')

accountName: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/accountName')

accountServiceHomepage: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/accountServiceHomepage')

age: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/age')

aimChatID: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/aimChatID')

based_near: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/based_near')

birthday: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/birthday')

currentProject: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/currentProject')

depiction: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/depiction')

depicts: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/depicts')

dnaChecksum: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/dnaChecksum')

familyName: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/familyName')

family_name: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/family_name')

firstName: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/firstName')

focus: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/focus')

fundedBy: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/fundedBy')

geekcode: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/geekcode')

gender: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/gender')

givenName: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/givenName')

givenname: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/givenname')

holdsAccount: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/holdsAccount')

homepage: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/homepage')

icqChatID: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/icqChatID')

img: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/img')

interest: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/interest')

isPrimaryTopicOf: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/isPrimaryTopicOf')

jabberID: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/jabberID')

knows: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/knows')

lastName: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/lastName')

logo: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/logo')

made: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/made')

maker: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/maker')

mbox: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/mbox')

mbox_sha1sum: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/mbox_sha1sum')

member: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/member')

membershipClass: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/membershipClass')

msnChatID: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/msnChatID')

myersBriggs: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/myersBriggs')

name: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/name')

nick: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/nick')

openid: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/openid')

page: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/page')

pastProject: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/pastProject')

phone: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/phone')

plan: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/plan')

primaryTopic: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/primaryTopic')

publications: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/publications')

schoolHomepage: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/schoolHomepage')

sha1: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/sha1')

skypeID: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/skypeID')

status: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/status')

surname: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/surname')

theme: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/theme')

thumbnail: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/thumbnail')

tipjar: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/tipjar')

title: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/title')

topic: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/topic')

topic_interest: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/topic_interest')

weblog: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/weblog')

workInfoHomepage: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/workInfoHomepage')

workplaceHomepage: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/workplaceHomepage')

yahooChatID: *URIRef* = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/yahooChatID')

class rdflib.**Graph**(*store='default'*, *identifier=None*, *namespace_manager=None*, *base=None*, *bind_namespaces='core'*)

Bases: *Node*

An RDF Graph

The constructor accepts one argument, the "store" that will be used to store the graph data (see the "store" package for stores currently shipped with rdflib).

Stores can be context-aware or unaware. Unaware stores take up (some) less space but cannot support features that require context, such as true merging/demerging of sub-graphs and provenance.

Even if used with a context-aware store, Graph will only expose the quads which belong to the default graph. To access the rest of the data, *ConjunctiveGraph* or *Dataset* classes can be used instead.

The Graph constructor can take an identifier which identifies the Graph by name. If none is given, the graph is assigned a BNode for its identifier.

For more on named graphs, see: http://www.w3.org/2004/03/trix/

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **namespace_manager** (Optional[*NamespaceManager*]) –
- **base** (Optional[str]) –
- **bind_namespaces** (Literal['core', 'rdflib', 'none']) –

__add__(*other*)

Set-theoretic union BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__and__(*other*)

Set-theoretic intersection. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__annotations__ = {'__identifier': '_ContextIdentifierType', '__store': 'Store'}

__cmp__(*other*)

Return type
int

__contains__(*triple*)

Support for 'triple in graph' syntax

Parameters
triple (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type
bool

__dict__ = mappingproxy({'__module__': 'rdflib.graph', '__doc__': 'An RDF Graph\n\n The constructor accepts one argument, the "store"\n that will be used to store the graph data (see the "store"\n package for stores currently shipped with rdflib).\n\n Stores can be context-aware or unaware. Unaware stores take up\n (some) less space but cannot support features that require\n context, such as true merging/demerging of sub-graphs and\n provenance.\n\n Even if used with a context-aware store, Graph will only expose the quads which\n belong to the default graph. To access the rest of the data, `ConjunctiveGraph` or\n `Dataset` classes can be used instead.\n\n The Graph constructor can take an identifier which identifies the Graph\n by name. If none is given, the graph is assigned a BNode for its\n identifier.\n\n For more on named graphs, see: http://www.w3.org/2004/03/trix/\n ', '__init__': <function Graph.__init__>, 'store': <property object>, 'identifier': <property object>, 'namespace_manager': <property object>, '__repr__': <function Graph.__repr__>, '__str__': <function Graph.__str__>, 'toPython': <function Graph.toPython>, 'destroy': <function Graph.destroy>, 'commit': <function Graph.commit>, 'rollback': <function Graph.rollback>, 'open': <function Graph.open>, 'close': <function Graph.close>, 'add': <function Graph.add>, 'addN': <function Graph.addN>, 'remove': <function Graph.remove>, 'triples': <function Graph.triples>, '__getitem__': <function Graph.__getitem__>, '__len__': <function Graph.__len__>, '__iter__': <function Graph.__iter__>, '__contains__': <function Graph.__contains__>, '__hash__': <function Graph.__hash__>, '__cmp__': <function Graph.__cmp__>, '__eq__': <function Graph.__eq__>, '__lt__': <function Graph.__lt__>, '__le__': <function Graph.__le__>, '__gt__': <function Graph.__gt__>, '__ge__': <function Graph.__ge__>, '__iadd__': <function Graph.__iadd__>, '__isub__': <function Graph.__isub__>, '__add__': <function Graph.__add__>, '__mul__': <function Graph.__mul__>, '__sub__': <function Graph.__sub__>, '__xor__': <function Graph.__xor__>, '__or__': <function Graph.__add__>, '__and__': <function Graph.__mul__>, 'set': <function Graph.set>, 'subjects': <function Graph.subjects>, 'predicates': <function Graph.predicates>, 'objects': <function Graph.objects>, 'subject_predicates': <function Graph.subject_predicates>, 'subject_objects': <function Graph.subject_objects>, 'predicate_objects': <function Graph.predicate_objects>, 'triples_choices': <function Graph.triples_choices>, 'value': <function Graph.value>, 'items': <function Graph.items>, 'transitiveClosure': <function Graph.transitiveClosure>, 'transitive_objects': <function Graph.transitive_objects>, 'transitive_subjects': <function Graph.transitive_subjects>, 'qname': <function Graph.qname>, 'compute_qname': <function Graph.compute_qname>, 'bind': <function Graph.bind>, 'namespaces': <function Graph.namespaces>, 'absolutize': <function Graph.absolutize>, 'serialize': <function Graph.serialize>, 'print': <function Graph.print>, 'parse': <function Graph.parse>, 'query': <function Graph.query>, 'update': <function Graph.update>, 'n3': <function Graph.n3>, '__reduce__': <function Graph.__reduce__>, 'isomorphic': <function Graph.isomorphic>, 'connected': <function Graph.connected>, 'all_nodes': <function Graph.all_nodes>, 'collection': <function Graph.collection>, 'resource': <function Graph.resource>, '_process_skolem_tuples': <function Graph._process_skolem_tuples>, 'skolemize': <function Graph.skolemize>, 'de_skolemize': <function Graph.de_skolemize>, 'cbd': <function Graph.cbd>, '__dict__': <attribute '__dict__' of 'Graph' objects>, '__weakref__': <attribute '__weakref__' of 'Graph' objects>, '__annotations__': {'__identifier': '_ContextIdentifierType', '__store': 'Store'}})

__eq__(*other*)

Return self==value.

Return type

bool

__ge__(*other*)

Return self>=value.

Parameters
other (*Graph*) –

Return type
bool

__getitem__(*item*)

A graph can be "sliced" as a shortcut for the triples method The python slice syntax is (ab)used for specifying triples. A generator over matches is returned, the returned tuples include only the parts not given

```
>>> import rdflib
>>> g = rdflib.Graph()
>>> g.add((rdflib.URIRef("urn:bob"), namespace.RDFS.label, rdflib.Literal("Bob
↪")))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
```

```
>>> list(g[rdflib.URIRef("urn:bob")]) # all triples about bob
[(rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#label'), rdflib.term.
↪Literal('Bob'))]
```

```
>>> list(g[:namespace.RDFS.label]) # all label triples
[(rdflib.term.URIRef('urn:bob'), rdflib.term.Literal('Bob'))]
```

```
>>> list(g[::rdflib.Literal("Bob")]) # all triples with bob as object
[(rdflib.term.URIRef('urn:bob'), rdflib.term.URIRef('http://www.w3.org/2000/01/
↪rdf-schema#label'))]
```

Combined with SPARQL paths, more complex queries can be written concisely:

Name of all Bobs friends:

g[bob : FOAF.knows/FOAF.name]

Some label for Bob:

g[bob : DC.title|FOAF.name|RDFS.label]

All friends and friends of friends of Bob

g[bob : FOAF.knows * "+"]

etc.

New in version 4.0.

__gt__(*other*)

Return self>value.

Return type
bool

__hash__()

Return hash(self).

Return type
int

__iadd__(*other*)

Add all triples in Graph other to Graph. BNode IDs are not changed.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **other** (Iterable[Tuple[*Node*, *Node*, *Node*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

__init__(*store='default'*, *identifier=None*, *namespace_manager=None*, *base=None*, *bind_namespaces='core'*)

Parameters

- **store** (Union[*Store*, str]) –
- **identifier** (Union[*IdentifiedNode*, str, None]) –
- **namespace_manager** (Optional[*NamespaceManager*]) –
- **base** (Optional[str]) –
- **bind_namespaces** (Literal['core', 'rdflib', 'none']) –

__isub__(*other*)

Subtract all triples in Graph other from Graph. BNode IDs are not changed.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **other** (Iterable[Tuple[*Node*, *Node*, *Node*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

__iter__()

Iterates over all triples in the store

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

__le__(*other*)

Return self<=value.

Parameters
other (*Graph*) –

Return type
bool

__len__()

Returns the number of triples in the graph

If context is specified then the number of triples in the context is returned instead.

Return type
int

__lt__(*other*)

Return self<value.

Return type
bool

__module__ = 'rdflib.graph'

__mul__(*other*)

Set-theoretic intersection. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__or__(*other*)

Set-theoretic union BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Graph*], Tuple[*Store*, *IdentifiedNode*]]

__repr__()

Return repr(self).

Return type
str

__str__()

Return str(self).

Return type
str

__sub__(*other*)

Set-theoretic difference. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

__weakref__

list of weak references to the object (if defined)

__xor__(*other*)

Set-theoretic XOR. BNode IDs are not changed.

Parameters
other (*Graph*) –

Return type
Graph

absolutize(*uri*, *defrag=1*)

Turn uri into an absolute URI if it's not one already

Parameters

- **uri** (str) –
- **defrag** (int) –

Return type
URIRef

add(*triple*)

Add a triple with self as context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[*Node*, *Node*, *Node*]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

addN(*quads*)

Add a sequence of triple with context

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **quads** (Iterable[Tuple[*Node*, *Node*, *Node*, *Graph*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

all_nodes()

Return type
Set[*Node*]

bind(*prefix*, *namespace*, *override=True*, *replace=False*)

Bind prefix to namespace

If override is True will bind namespace to given prefix even if namespace was already bound to a different prefix.

if replace, replace any existing prefix with the new namespace

for example: graph.bind("foaf", "http://xmlns.com/foaf/0.1/")

Parameters

- **prefix** (Optional[str]) –
- **namespace** (Any) –
- **override** (bool) –
- **replace** (bool) –

Return type
None

cbd(*resource*)

Retrieves the Concise Bounded Description of a Resource from a Graph

Concise Bounded Description (CBD) is defined in [1] as:

Given a particular node (the starting node) in a particular RDF graph (the source graph), a subgraph of that particular graph, taken to comprise a concise bounded description of the resource denoted by the starting node, can be identified as follows:

1. **Include in the subgraph all statements in the source graph where the subject of the statement is the**
   starting node;
2. **Recursively, for all statements identified in the subgraph thus far having a blank node object, include**
   in the subgraph all statements in the source graph where the subject of the statement is the blank node in question and which are not already included in the subgraph.
3. **Recursively, for all statements included in the subgraph thus far, for all reifications of each statement**
   in the source graph, include the concise bounded description beginning from the rdf:Statement node of each reification.

This results in a subgraph where the object nodes are either URI references, literals, or blank nodes not serving as the subject of any statement in the graph.

[1] https://www.w3.org/Submission/CBD/

Parameters
resource (*Node*) – a URIRef object, of the Resource for queried for

Return type
Graph

Returns
a Graph, subgraph of self

close(*commit_pending_transaction=False*)

Close the graph store

Might be necessary for stores that require closing a connection to a database or releasing some resource.

Parameters
commit_pending_transaction (bool) –

Return type
None

collection(*identifier*)

Create a new `Collection` instance.

Parameters:

- `identifier`: a URIRef or BNode instance.

Example:

```
>>> graph = Graph()
>>> uri = URIRef("http://example.org/resource")
>>> collection = graph.collection(uri)
>>> assert isinstance(collection, Collection)
>>> assert collection.uri is uri
```

(continues on next page)

(continued from previous page)

```
>>> assert collection.graph is graph
>>> collection += [ Literal(1), Literal(2) ]
```

Parameters
identifier (*Node*) –

Return type
Collection

commit()

Commits active transactions

Parameters
self (TypeVar(_GraphT, bound= *Graph*)) –

Return type
TypeVar(_GraphT, bound= *Graph*)

compute_qname(*uri*, *generate=True*)

Parameters

- **uri** (str) –
- **generate** (bool) –

Return type
Tuple[str, *URIRef*, str]

connected()

Check if the Graph is connected

The Graph is considered undirectional.

Performs a search on the Graph, starting from a random node. Then iteratively goes depth-first through the triplets where the node is subject and object. Return True if all nodes have been visited and False if it cannot continue and there are still unvisited nodes left.

Return type
bool

de_skolemize(*new_graph=None*, *uriref=None*)

Parameters

- **new_graph** (Optional[*Graph*]) –
- **uriref** (Optional[*URIRef*]) –

Return type
Graph

destroy(*configuration*)

Destroy the store identified by `configuration` if supported

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **configuration** (str) –

Return type
TypeVar(_GraphT, bound= *Graph*)

property identifier: *IdentifiedNode*

Return type
IdentifiedNode

isomorphic(*other*)

does a very basic check if these graphs are the same If no BNodes are involved, this is accurate.

See rdflib.compare for a correct implementation of isomorphism checks

Parameters
other (*Graph*) –

Return type
bool

items(*list*)

Generator over all items in the resource specified by list

list is an RDF collection.

Parameters
list (*Node*) –

Return type
Generator[*Node*, None, None]

n3()

Return an n3 identifier for the Graph

Return type
str

property namespace_manager: *NamespaceManager*

this graph's namespace-manager

Return type
NamespaceManager

namespaces()

Generator over all the prefix, namespace tuples

Return type
Generator[Tuple[str, *URIRef*], None, None]

objects(*subject=None*, *predicate=None*, *unique=False*)

A generator of (optionally unique) objects with the given subject and predicate

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Union[None, *Path*, *Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

open(*configuration*, *create=False*)

Open the graph store

Might be necessary for stores that require opening a connection to a database or acquiring some resource.

Parameters

- **configuration** (str) –
- **create** (bool) –

Return type
Optional[int]

parse(*source=None*, *publicID=None*, *format=None*, *location=None*, *file=None*, *data=None*, ***args*)

Parse an RDF source adding the resulting triples to the Graph.

The source is specified using one of source, location, file or data.

> **Caution:** This method can access directly or indirectly requested network or file resources, for example, when parsing JSON-LD documents with `@context` directives that point to a network location.
>
> When processing untrusted or potentially malicious documents, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- `source`: An InputSource, file-like object, or string. In the case of a string the string is the location of the source.
- `location`: A string indicating the relative or absolute URL of the source. Graph's absolutize method is used if a relative location is specified.
- `file`: A file-like object.
- `data`: A string containing the data to be parsed.
- `format`: Used if format can not be determined from source, e.g. file extension or Media Type. Defaults to text/turtle. Format support can be extended with plugins, but "xml", "n3" (use for turtle), "nt" & "trix" are built in.
- `publicID`: the logical URI to use as the document base. If None specified the document location is used (at least in the case where there is a document location).

Returns

- self, the graph instance.

Examples:

```
>>> my_data = '''
... <rdf:RDF
...   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
...   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
... >
...   <rdf:Description>
...     <rdfs:label>Example</rdfs:label>
```

(continues on next page)

(continued from previous page)

```
...     <rdfs:comment>This is really just an example.</rdfs:comment>
...   </rdf:Description>
... </rdf:RDF>
... '''
>>> import os, tempfile
>>> fd, file_name = tempfile.mkstemp()
>>> f = os.fdopen(fd, "w")
>>> dummy = f.write(my_data)  # Returns num bytes written
>>> f.close()
```

```
>>> g = Graph()
>>> result = g.parse(data=my_data, format="application/rdf+xml")
>>> len(g)
2
```

```
>>> g = Graph()
>>> result = g.parse(location=file_name, format="application/rdf+xml")
>>> len(g)
2
```

```
>>> g = Graph()
>>> with open(file_name, "r") as f:
...     result = g.parse(f, format="application/rdf+xml")
>>> len(g)
2
```

```
>>> os.remove(file_name)
```

```
>>> # default turtle parsing
>>> result = g.parse(data="<http://example.com/a> <http://example.com/a> <http:/
↪/example.com/a> .")
>>> len(g)
3
```

Parameters

- **source** (Union[IO[bytes], TextIO, *InputSource*, str, bytes, PurePath, None]) –
- **publicID** (Optional[str]) –
- **format** (Optional[str]) –
- **location** (Optional[str]) –
- **file** (Union[BinaryIO, TextIO, None]) –
- **data** (Union[str, bytes, None]) –
- **args** (Any) –

Return type

Graph

predicate_objects(*subject=None*, *unique=False*)

A generator of (optionally unique) (predicate, object) tuples for the given subject

Parameters

- **subject** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

predicates(*subject=None*, *object=None*, *unique=False*)

A generator of (optionally unique) predicates with the given subject and object

Parameters

- **subject** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

print(*format='turtle'*, *encoding='utf-8'*, *out=None*)

Parameters

- **format** (str) –
- **encoding** (str) –
- **out** (Optional[TextIO]) –

Return type
None

qname(*uri*)

Parameters
uri (str) –

Return type
str

query(*query_object*, *processor='sparql'*, *result='sparql'*, *initNs=None*, *initBindings=None*, *use_store_provided=True*, ***kwargs*)

Query this graph.

A type of 'prepared queries' can be realised by providing initial variable bindings with initBindings

Initial namespaces are used to resolve prefixes used in the query, if none are given, the namespaces from the graph's namespace manager are used.

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in SERVICE directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Returntype
Result

Parameters

- **query_object** (Union[str, *Query*]) –
- **processor** (Union[str, *Processor*]) –
- **result** (Union[str, Type[*Result*]]) –
- **initNs** (Optional[Mapping[str, Any]]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –
- **use_store_provided** (bool) –
- **kwargs** (Any) –

Return type
Result

remove(*triple*)

Remove a triple from the graph

If the triple does not provide a context attribute, removes the triple from all contexts.

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

resource(*identifier*)

Create a new `Resource` instance.

Parameters:

- `identifier`: a URIRef or BNode instance.

Example:

```
>>> graph = Graph()
>>> uri = URIRef("http://example.org/resource")
>>> resource = graph.resource(uri)
>>> assert isinstance(resource, Resource)
>>> assert resource.identifier is uri
>>> assert resource.graph is graph
```

Parameters
identifier (Union[*Node*, str]) –

Return type
Resource

rollback()

Rollback active transactions

Parameters
self (TypeVar(_GraphT, bound= *Graph*)) –

Return type
TypeVar(_GraphT, bound= *Graph*)

serialize(*destination: None*, *format: str*, *base: Optional[str]*, *encoding: str*, ***args: Any*) → bytes

serialize(*destination: None = None*, *format: str = 'turtle'*, *base: Optional[str] = None*, ***, *encoding: str*, ***args: Any*) → bytes

serialize(*destination: None = None*, *format: str = 'turtle'*, *base: Optional[str] = None*, *encoding: None = None*, ***args: Any*) → str

serialize(*destination: Union[str, PurePath, IO[bytes]]*, *format: str = 'turtle'*, *base: Optional[str] = None*, *encoding: Optional[str] = None*, ***args: Any*) → *Graph*

serialize(*destination: Optional[Union[str, PurePath, IO[bytes]]] = None*, *format: str = 'turtle'*, *base: Optional[str] = None*, *encoding: Optional[str] = None*, ***args: Any*) → Union[bytes, str, *Graph*]

Serialize the graph.

Parameters

- **destination** (Union[str, PurePath, IO[bytes], None]) – The destination to serialize the graph to. This can be a path as a str or PurePath object, or it can be a IO[bytes] like object. If this parameter is not supplied the serialized graph will be returned.
- **format** (str) – The format that the output should be written in. This value references a *Serializer* plugin. Format support can be extended with plugins, but "xml", "n3", "turtle", "nt", "pretty-xml", "trix", "trig", "nquads", "json-ld" and "hext" are built in. Defaults to "turtle".
- **base** (Optional[str]) – The base IRI for formats that support it. For the turtle format this will be used as the @base directive.
- **encoding** (Optional[str]) – Encoding of output.
- **args** (Any) – Additional arguments to pass to the *Serializer* that will be used.
- **self** (TypeVar(_GraphT, bound= *Graph*)) –

Returns
The serialized graph if destination is None. The serialized graph is returned as str if no encoding is specified, and as bytes if an encoding is specified.

Return type
bytes if destination is None and encoding is not None.

Return type
str if destination is None and encoding is None.

Returns
self (i.e. the *Graph* instance) if destination is not None.

Return type
Graph if destination is not None.

set(*triple*)

Convenience method to update the value of object

Remove any existing triples for subject and predicate before adding (subject, predicate, object).

Parameters

- **self** (TypeVar(_GraphT, bound= *Graph*)) –
- **triple** (Tuple[*Node*, *Node*, *Node*]) –

Return type
TypeVar(_GraphT, bound= *Graph*)

skolemize(*new_graph=None*, *bnode=None*, *authority=None*, *basepath=None*)

Parameters

- **new_graph** (Optional[*Graph*]) –
- **bnode** (Optional[*BNode*]) –
- **authority** (Optional[str]) –
- **basepath** (Optional[str]) –

Return type
Graph

property store: *Store*

Return type
Store

subject_objects(*predicate=None*, *unique=False*)

A generator of (optionally unique) (subject, object) tuples for the given predicate

Parameters

- **predicate** (Union[None, *Path*, *Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subject_predicates(*object=None*, *unique=False*)

A generator of (optionally unique) (subject, predicate) tuples for the given object

Parameters

- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[Tuple[*Node*, *Node*], None, None]

subjects(*predicate=None*, *object=None*, *unique=False*)

A generator of (optionally unique) subjects with the given predicate and object

Parameters

- **predicate** (Union[None, *Path*, *Node*]) –
- **object** (Optional[*Node*]) –
- **unique** (bool) –

Return type
Generator[*Node*, None, None]

toPython()

Parameters
self (TypeVar(_GraphT, bound= *Graph*)) –

Return type
TypeVar(_GraphT, bound= *Graph*)

transitiveClosure(*func*, *arg*, *seen=None*)

Generates transitive closure of a user-defined function against the graph

```
>>> from rdflib.collection import Collection
>>> g=Graph()
>>> a=BNode("foo")
>>> b=BNode("bar")
>>> c=BNode("baz")
>>> g.add((a,RDF.first,RDF.type))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((a,RDF.rest,b))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((b,RDF.first,namespace.RDFS.label))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((b,RDF.rest,c))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((c,RDF.first,namespace.RDFS.comment))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> g.add((c,RDF.rest,RDF.nil))
<Graph identifier=... (<class 'rdflib.graph.Graph'>)>
>>> def topList(node,g):
...    for s in g.subjects(RDF.rest, node):
...       yield s
>>> def reverseList(node,g):
...    for f in g.objects(node, RDF.first):
...       print(f)
...    for s in g.subjects(RDF.rest, node):
...       yield s
```

```
>>> [rt for rt in g.transitiveClosure(
...     topList,RDF.nil)]
[rdflib.term.BNode('baz'),
 rdflib.term.BNode('bar'),
 rdflib.term.BNode('foo')]
```

```
>>> [rt for rt in g.transitiveClosure(
...     reverseList,RDF.nil)]
http://www.w3.org/2000/01/rdf-schema#comment
http://www.w3.org/2000/01/rdf-schema#label
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
[rdflib.term.BNode('baz'),
 rdflib.term.BNode('bar'),
 rdflib.term.BNode('foo')]
```

Parameters

- **func** (Callable[[TypeVar(_TCArgT), *Graph*], Iterable[TypeVar(_TCArgT)]]) –
- **arg** (TypeVar(_TCArgT)) –
- **seen** (Optional[Dict[TypeVar(_TCArgT), int]]) –

transitive_objects(*subject*, *predicate*, *remember=None*)

Transitively generate objects for the `predicate` relationship

Generated objects belong to the depth first transitive closure of the `predicate` relationship starting at `subject`.

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –
- **remember** (Optional[Dict[Optional[*Node*], int]]) –

Return type
Generator[Optional[*Node*], None, None]

transitive_subjects(*predicate*, *object*, *remember=None*)

Transitively generate subjects for the `predicate` relationship

Generated subjects belong to the depth first transitive closure of the `predicate` relationship starting at `object`.

Parameters

- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –
- **remember** (Optional[Dict[Optional[*Node*], int]]) –

Return type
Generator[Optional[*Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Optional[*Node*], Optional[*Node*]]*) → Generator[Tuple[*Node*, *Node*, *Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Path, Optional[*Node*]]*) → Generator[Tuple[*Node*, *Path*, *Node*], None, None]

triples(*triple: Tuple[Optional[*Node*], Optional[Union[Path, *Node*]], Optional[*Node*]]*) → Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

Generator over the triple store

Returns triples that match the given triple pattern. If triple pattern does not provide a context, all contexts will be searched.

Parameters
triple (Tuple[Optional[*Node*], Union[*Path*, *Node*, None], Optional[*Node*]]) –

Return type
Generator[Union[Tuple[*Node*, *Node*, *Node*], Tuple[*Node*, *Path*, *Node*]], None, None]

triples_choices(*triple*, *context=None*)

Parameters

- **triple** (Union[Tuple[List[*Node*], *Node*, *Node*], Tuple[*Node*, List[*Node*], *Node*], Tuple[*Node*, *Node*, List[*Node*]]]) –
- **context** (Optional[*Graph*]) –

Return type
Generator[Tuple[*Node*, *Node*, *Node*], None, None]

update(*update_object*, *processor='sparql'*, *initNs=None*, *initBindings=None*, *use_store_provided=True*, ***kwargs*)

Update this graph with the given update query.

> **Caution:** This method can access indirectly requested network endpoints, for example, query processing will attempt to access network endpoints specified in `SERVICE` directives.
>
> When processing untrusted or potentially malicious queries, measures should be taken to restrict network and file access.
>
> For information on available security measures, see the RDFLib *Security Considerations* documentation.

Parameters

- **update_object** (Union[*Update*, str]) –
- **processor** (Union[str, *UpdateProcessor*]) –
- **initNs** (Optional[Mapping[str, Any]]) –
- **initBindings** (Optional[Mapping[str, *Identifier*]]) –
- **use_store_provided** (bool) –
- **kwargs** (Any) –

Return type
None

value(*subject: None = None*, *predicate: None = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: Optional[Node] = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

value(*subject: Optional[Node] = None*, *predicate: None = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: None = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

value(*subject: None = None*, *predicate: Optional[Node] = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: None = None*, *default: Optional[Node] = None*, *any: bool = True*) → None

value(*subject: Optional[Node] = None*, *predicate: Optional[Node] = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')*, *object: Optional[Node] = None*, *default: Optional[Node] = None*, *any: bool = True*) → Optional[*Node*]

Get a value for a pair of two criteria

Exactly one of subject, predicate, object must be None. Useful if one knows that there may only be one value.

It is one of those situations that occur a lot, hence this 'macro' like utility

Parameters: subject, predicate, object – exactly one must be None default – value to be returned if no values found any – if True, return any value in the case there is more than one, else, raise UniquenessError

Parameters

- **subject** (Optional[*Node*]) –
- **predicate** (Optional[*Node*]) –
- **object** (Optional[*Node*]) –

- **default** (Optional[*Node*]) –
- **any** (bool) –

Return type
Optional[*Node*]

class rdflib.**IdentifiedNode**(*value: str*)

Bases: *Identifier*

An abstract class, primarily defined to identify Nodes that are not Literals.

The name "Identified Node" is not explicitly defined in the RDF specification, but can be drawn from this section: https://www.w3.org/TR/rdf-concepts/#section-URI-Vocabulary

__annotations__ = {}

__dict__ = mappingproxy({'__module__': 'rdflib.term', '__doc__': '\n An abstract class, primarily defined to identify Nodes that are not Literals.\n\n The name "Identified Node" is not explicitly defined in the RDF specification, but can be drawn from this section:\n https://www.w3.org/TR/rdf-concepts/#section-URI-Vocabulary\n ', '__getnewargs__': <function IdentifiedNode.__getnewargs__>, 'toPython': <function IdentifiedNode.toPython>, '__dict__': <attribute '__dict__' of 'IdentifiedNode' objects>, '__weakref__': <attribute '__weakref__' of 'IdentifiedNode' objects>, '__annotations__': {}})

__getnewargs__()

Return type
Tuple[str]

__module__ = 'rdflib.term'

__weakref__
list of weak references to the object (if defined)

toPython()

Return type
str

class rdflib.**Literal**(*lexical_or_value: Any*, *lang: Optional[str] = None*, *datatype: Optional[str] = None*, *normalize: Optional[bool] = None*)

Bases: *Identifier*

RDF 1.1's Literals Section: http://www.w3.org/TR/rdf-concepts/#section-Graph-Literal

Literals are used for values such as strings, numbers, and dates.

A literal in an RDF graph consists of two or three elements:

- a lexical form, being a Unicode string, which SHOULD be in Normal Form C
- a datatype IRI, being an IRI identifying a datatype that determines how the lexical form maps to a literal value, and
- if and only if the datatype IRI is `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`, a non-empty language tag. The language tag MUST be well-formed according to section 2.2.9 of Tags for identifying languages.

A literal is a language-tagged string if the third element is present. Lexical representations of language tags MAY be converted to lower case. The value space of language tags is always in lower case.

—

For valid XSD datatypes, the lexical form is optionally normalized at construction time. Default behaviour is set by rdflib.NORMALIZE_LITERALS and can be overridden by the normalize parameter to __new__

Equality and hashing of Literals are done based on the lexical form, i.e.:

```
>>> from rdflib.namespace import XSD
```

```
>>> Literal('01') != Literal('1')  # clear - strings differ
True
```

but with data-type they get normalized:

```
>>> Literal('01', datatype=XSD.integer) != Literal('1', datatype=XSD.integer)
False
```

unless disabled:

```
>>> Literal('01', datatype=XSD.integer, normalize=False) != Literal('1',␣
↪datatype=XSD.integer)
True
```

Value based comparison is possible:

```
>>> Literal('01', datatype=XSD.integer).eq(Literal('1', datatype=XSD.float))
True
```

The eq method also provides limited support for basic python types:

```
>>> Literal(1).eq(1) # fine - int compatible with xsd:integer
True
>>> Literal('a').eq('b') # fine - str compatible with plain-lit
False
>>> Literal('a', datatype=XSD.string).eq('a') # fine - str compatible with␣
↪xsd:string
True
>>> Literal('a').eq(1) # not fine, int incompatible with plain-lit
NotImplemented
```

Greater-than/less-than ordering comparisons are also done in value space, when compatible datatypes are used. Incompatible datatypes are ordered by DT, or by lang-tag. For other nodes the ordering is None < BNode < URIRef < Literal

Any comparison with non-rdflib Node are “NotImplemented” In PY3 this is an error.

```
>>> from rdflib import Literal, XSD
>>> lit2006 = Literal('2006-01-01',datatype=XSD.date)
>>> lit2006.toPython()
datetime.date(2006, 1, 1)
>>> lit2006 < Literal('2007-01-01',datatype=XSD.date)
True
>>> Literal(datetime.utcnow()).datatype
```

(continues on next page)

(continued from previous page)

```
rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#dateTime')
>>> Literal(1) > Literal(2) # by value
False
>>> Literal(1) > Literal(2.0) # by value
False
>>> Literal('1') > Literal(1) # by DT
True
>>> Literal('1') < Literal('1') # by lexical form
False
>>> Literal('a', lang='en') > Literal('a', lang='fr') # by lang-tag
False
>>> Literal(1) > URIRef('foo') # by node-type
True
```

The > < operators will eat this NotImplemented and throw a TypeError (py3k):

```
>>> Literal(1).__gt__(2.0)
NotImplemented
```

__abs__()

```
>>> abs(Literal(-1))
rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> from rdflib.namespace import XSD
>>> abs( Literal("-1", datatype=XSD.integer))
rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> abs(Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
```

Return type
Literal

__add__(*val*)

```
>>> from rdflib.namespace import XSD
>>> Literal(1) + 1
rdflib.term.Literal(u'2', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> Literal("1") + "1"
rdflib.term.Literal(u'11')
```

Handling dateTime/date/time based operations in Literals >>> a = Literal('2006-01-01T20:50:00', datatype=XSD.dateTime) >>> b = Literal('P31D', datatype=XSD.duration) >>> (a + b) rdflib.term.Literal('2006-02-01T20:50:00', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dateTime')) >>> from rdflib.namespace import XSD >>> a = Literal('2006-07-01T20:52:00', datatype=XSD.dateTime) >>> b = Literal('P122DT15H58M', datatype=XSD.duration)

>>> (a + b) rdflib.term.Literal('2006-11-01T12:50:00', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dateTime'))

Parameters
val (Any) –

Return type
Literal

__annotations__ = {'_datatype': typing.Optional[rdflib.term.URIRef], '_ill_typed': typing.Optional[bool], '_language': typing.Optional[str], '_value': typing.Any}

__bool__()

Is the Literal "True" This is used for if statements, bool(literal), etc.

Return type
bool

__eq__(*other*)

Literals are only equal to other literals.

"Two literals are equal if and only if all of the following hold: * The strings of the two lexical forms compare equal, character by character. * Either both or neither have language tags. * The language tags, if any, compare equal. * Either both or neither have datatype URIs. * The two datatype URIs, if any, compare equal, character by character." – 6.5.1 Literal Equality (RDF: Concepts and Abstract Syntax)

```
>>> Literal("1", datatype=URIRef("foo")) == Literal("1", datatype=URIRef("foo"))
True
>>> Literal("1", datatype=URIRef("foo")) == Literal("1", datatype=URIRef("foo2
↪"))
False
```

```
>>> Literal("1", datatype=URIRef("foo")) == Literal("2", datatype=URIRef("foo"))
False
>>> Literal("1", datatype=URIRef("foo")) == "asdf"
False
>>> from rdflib import XSD
>>> Literal('2007-01-01', datatype=XSD.date) == Literal('2007-01-01',␣
↪datatype=XSD.date)
True
>>> Literal('2007-01-01', datatype=XSD.date) == date(2007, 1, 1)
False
>>> Literal("one", lang="en") == Literal("one", lang="en")
True
>>> Literal("hast", lang='en') == Literal("hast", lang='de')
False
>>> Literal("1", datatype=XSD.integer) == Literal(1)
True
>>> Literal("1", datatype=XSD.integer) == Literal("01", datatype=XSD.integer)
True
```

Parameters
other (Any) –

Return type
bool

__ge__(*other*)

Return self>=value.

Parameters
other (Any) –

Return type
bool

__getstate__()

Return type
Tuple[None, Dict[str, Optional[str]]]

__gt__(*other*)

This implements ordering for Literals, the other comparison methods delegate here

This tries to implement this: http://www.w3.org/TR/sparql11-query/#modOrderBy

In short, Literals with compatible data-types are ordered in value space, i.e. >>> from rdflib import XSD

```
>>> Literal(1) > Literal(2) # int/int
False
>>> Literal(2.0) > Literal(1) # double/int
True
>>> from decimal import Decimal
>>> Literal(Decimal("3.3")) > Literal(2.0) # decimal/double
True
>>> Literal(Decimal("3.3")) < Literal(4.0) # decimal/double
True
>>> Literal('b') > Literal('a') # plain lit/plain lit
True
>>> Literal('b') > Literal('a', datatype=XSD.string) # plain lit/xsd:str
True
```

Incompatible datatype mismatches ordered by DT

```
>>> Literal(1) > Literal("2") # int>string
False
```

Langtagged literals by lang tag >>> Literal("a", lang="en") > Literal("a", lang="fr") False

Parameters
other (Any) –

Return type
bool

__hash__()

```
>>> from rdflib.namespace import XSD
>>> a = {Literal('1', datatype=XSD.integer):'one'}
>>> Literal('1', datatype=XSD.double) in a
False
```

"Called for the key object for dictionary operations, and by the built-in function hash(). Should return a 32-bit integer usable as a hash value for dictionary operations. The only required property is that objects which compare equal have the same hash value; it is advised to somehow mix together (e.g., using exclusive

or) the hash values for the components of the object that also play a part in comparison of objects.” – 3.4.1 Basic customization (Python)

“Two literals are equal if and only if all of the following hold: * The strings of the two lexical forms compare equal, character by character. * Either both or neither have language tags. * The language tags, if any, compare equal. * Either both or neither have datatype URIs. * The two datatype URIs, if any, compare equal, character by character.” – 6.5.1 Literal Equality (RDF: Concepts and Abstract Syntax)

Return type
int

__invert__()

```
>>> ~(Literal(-1))
rdflib.term.Literal(u'0', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> from rdflib.namespace import XSD
>>> ~( Literal("-1", datatype=XSD.integer))
rdflib.term.Literal(u'0', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

Not working:

```
>>> ~(Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
```

Return type
Literal

__le__(*other*)

```
>>> from rdflib.namespace import XSD
>>> Literal('2007-01-01T10:00:00', datatype=XSD.dateTime
...     ) <= Literal('2007-01-01T10:00:00', datatype=XSD.dateTime)
True
```

Parameters
other (Any) –

Return type
bool

__lt__(*other*)

Return self<value.

Parameters
other (Any) –

Return type
bool

__module__ = 'rdflib.term'

__neg__()

```
>>> (- Literal(1))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> (- Literal(10.5))
rdflib.term.Literal(u'-10.5', datatype=rdflib.term.URIRef(u'http://www.w3.org/
↪2001/XMLSchema#double'))
>>> from rdflib.namespace import XSD
>>> (- Literal("1", datatype=XSD.integer))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> (- Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
>>>
```

Return type
Literal

static __new__(*cls*, *lexical_or_value*, *lang=None*, *datatype=None*, *normalize=None*)

Parameters

- **lexical_or_value** (Any) –
- **lang** (Optional[str]) –
- **datatype** (Optional[str]) –
- **normalize** (Optional[bool]) –

Return type
Literal

__pos__()

```
>>> (+ Literal(1))
rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> (+ Literal(-1))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> from rdflib.namespace import XSD
>>> (+ Literal("-1", datatype=XSD.integer))
rdflib.term.Literal(u'-1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/
↪XMLSchema#integer'))
```

```
>>> (+ Literal("1"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Not a number; rdflib.term.Literal(u'1')
```

Return type
Literal

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Literal*], Tuple[str, Optional[str], Optional[str]]]

__repr__()

Return repr(self).

Return type
str

__setstate__(*arg*)

Parameters
arg (Tuple[Any, Dict[str, Any]]) –

Return type
None

__slots__ = ('_language', '_datatype', '_value', '_ill_typed')

__sub__(*val*)

```
>>> from rdflib.namespace import XSD
>>> Literal(2) - 1
rdflib.term.Literal('1', datatype=rdflib.term.URIRef('http://www.w3.org/2001/
↪XMLSchema#integer'))
>>> Literal(1.1) - 1.0
rdflib.term.Literal('0.10000000000000009', datatype=rdflib.term.URIRef('http://
↪www.w3.org/2001/XMLSchema#double'))
>>> Literal(1.1) - 1
rdflib.term.Literal('0.1', datatype=rdflib.term.URIRef('http://www.w3.org/2001/
↪XMLSchema#decimal'))
>>> Literal(1.1, datatype=XSD.float) - Literal(1.0, datatype=XSD.float)
rdflib.term.Literal('0.10000000000000009', datatype=rdflib.term.URIRef('http://
↪www.w3.org/2001/XMLSchema#float'))
>>> Literal("1.1") - 1.0
Traceback (most recent call last):
...
TypeError: Not a number; rdflib.term.Literal('1.1')
>>> Literal(1.1, datatype=XSD.integer) - Literal(1.0, datatype=XSD.integer)
rdflib.term.Literal('0.10000000000000009', datatype=rdflib.term.URIRef('http://
↪www.w3.org/2001/XMLSchema#integer'))
```

Handling dateTime/date/time based operations in Literals >>> a = Literal('2006-01-01T20:50:00', datatype=XSD.dateTime) >>> b = Literal('2006-02-01T20:50:00', datatype=XSD.dateTime) >>> (b - a) rdflib.term.Literal('P31D', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration')) >>> from rdflib.namespace import XSD >>> a = Literal('2006-07-01T20:52:00', datatype=XSD.dateTime) >>> b = Literal('2006-11-01T12:50:00', datatype=XSD.dateTime) >>> (a - b) rdflib.term.Literal('-P122DT15H58M', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration')) >>> (b - a) rdflib.term.Literal('P122DT15H58M', datatype=rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration'))

Parameters
val (Any) –

Return type
Literal

property datatype: Optional[URIRef]

Return type
Optional[*URIRef*]

eq(*other*)

Compare the value of this literal with something else

Either, with the value of another literal comparisons are then done in literal "value space", and according to the rules of XSD subtype-substitution/type-promotion

OR, with a python object:

basestring objects can be compared with plain-literals, or those with datatype xsd:string

bool objects with xsd:boolean

a int, long or float with numeric xsd types

isodate date,time,datetime objects with xsd:date,xsd:time or xsd:datetime

Any other operations returns NotImplemented

Parameters
other (Any) –

Return type
bool

property ill_typed: Optional[bool]

For recognized datatype IRIs, this value will be True if the literal is ill formed, otherwise it will be False. *Literal.value* (i.e. the literal value) should always be defined if this property is False, but should not be considered reliable if this property is True.

If the literal's datatype is None or not in the set of recognized datatype IRIs this value will be None.

Return type
Optional[bool]

property language: Optional[str]

Return type
Optional[str]

n3(*namespace_manager=None*)

Returns a representation in the N3 format.

Examples:

```
>>> Literal("foo").n3()
u'"foo"'
```

Strings with newlines or triple-quotes:

```
>>> Literal("foo\nbar").n3()
u'"""foo\nbar"""'

>>> Literal("''\'").n3()
u'"\'\'\'"'

>>> Literal('"""').n3()
u'"\\"\\"\\""'
```

Language:

```
>>> Literal("hello", lang="en").n3()
u'"hello"@en'
```

Datatypes:

```
>>> Literal(1).n3()
u'"1"^^<http://www.w3.org/2001/XMLSchema#integer>'

>>> Literal(1.0).n3()
u'"1.0"^^<http://www.w3.org/2001/XMLSchema#double>'

>>> Literal(True).n3()
u'"true"^^<http://www.w3.org/2001/XMLSchema#boolean>'
```

Datatype and language isn't allowed (datatype takes precedence):

```
>>> Literal(1, lang="en").n3()
u'"1"^^<http://www.w3.org/2001/XMLSchema#integer>'
```

Custom datatype:

```
>>> footype = URIRef("http://example.org/ns#foo")
>>> Literal("1", datatype=footype).n3()
u'"1"^^<http://example.org/ns#foo>'
```

Passing a namespace-manager will use it to abbreviate datatype URIs:

```
>>> from rdflib import Graph
>>> Literal(1).n3(Graph().namespace_manager)
u'"1"^^xsd:integer'
```

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

neq(*other*)

A "semantic"/interpreted not equal function, by default, same as __ne__

Parameters
other (Any) –

Return type
bool

normalize()

Returns a new literal with a normalised lexical representation of this literal >>> from rdflib import XSD >>> Literal("01", datatype=XSD.integer, normalize=False).normalize() rdflib.term.Literal(u'1', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#integer'))

Illegal lexical forms for the datatype given are simply passed on >>> Literal("a", datatype=XSD.integer, normalize=False) rdflib.term.Literal(u'a', datatype=rdflib.term.URIRef(u'http://www.w3.org/2001/XMLSchema#integer'))

Return type
Literal

toPython()

Returns an appropriate python datatype derived from this RDF Literal

Return type
Any

property value: Any

Return type
Any

class rdflib.**Namespace**(*value: Union[str, bytes]*)

Bases: str

Utility class for quickly generating URIRefs with a common prefix

```
>>> from rdflib.namespace import Namespace
>>> n = Namespace("http://example.org/")
>>> n.Person # as attribute
rdflib.term.URIRef('http://example.org/Person')
>>> n['first-name'] # as item - for things that are not valid python identifiers
rdflib.term.URIRef('http://example.org/first-name')
>>> n.Person in n
True
>>> n2 = Namespace("http://example2.org/")
>>> n.Person in n2
False
```

__contains__(*ref*)

Allows to check if a URI is within (starts with) this Namespace.

```
>>> from rdflib import URIRef
>>> namespace = Namespace('http://example.org/')
>>> uri = URIRef('http://example.org/foo')
>>> uri in namespace
True
>>> person_class = namespace['Person']
>>> person_class in namespace
True
>>> obj = URIRef('http://not.example.org/bar')
>>> obj in namespace
False
```

Parameters
ref (str) –

Return type
bool

__dict__ = mappingproxy({'__module__': 'rdflib.namespace', '__doc__': '\n Utility class for quickly generating URIRefs with a common prefix\n\n >>> from rdflib.namespace import Namespace\n >>> n = Namespace("http://example.org/")\n >>> n.Person # as attribute\n rdflib.term.URIRef(\'http://example.org/Person\')\n >>> n[\'first-name\'] # as item - for things that are not valid python identifiers\n rdflib.term.URIRef(\'http://example.org/first-name\')\n >>> n.Person in n\n True\n >>> n2 = Namespace("http://example2.org/")\n >>> n.Person in n2\n False\n ', '__new__': <staticmethod object>, 'title': <property object>, 'term': <function Namespace.term>, '__getitem__': <function Namespace.__getitem__>, '__getattr__': <function Namespace.__getattr__>, '__repr__': <function Namespace.__repr__>, '__contains__': <function Namespace.__contains__>, '__dict__': <attribute '__dict__' of 'Namespace' objects>, '__weakref__': <attribute '__weakref__' of 'Namespace' objects>, '__annotations__': {}})

__getattr__(*name*)

Parameters
name (str) –

Return type
URIRef

__getitem__(*key*)

Return self[key].

Parameters
key (str) –

Return type
URIRef

__module__ = 'rdflib.namespace'

static __new__(*cls*, *value*)

Parameters
value (Union[str, bytes]) –

Return type
Namespace

__repr__()

Return repr(self).

Return type
str

__weakref__

list of weak references to the object (if defined)

term(*name*)

Parameters
name (str) –

Return type
URIRef

property title: *URIRef*

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

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

Return type
URIRef

class rdflib.**ODRL2**

Bases: *DefinedNamespace*

ODRL Version 2.2

The ODRL Vocabulary and Expression defines a set of concepts and terms (the vocabulary) and encoding mechanism (the expression) for permissions and obligations statements describing digital content usage based on the ODRL Information Model.

Generated from: https://www.w3.org/ns/odrl/2/ODRL22.ttl Date: 2020-05-26 14:20:02.352356

Action: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Action')

Agreement: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Agreement')

All: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/All')

All2ndConnections: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/All2ndConnections')

AllConnections: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/AllConnections')

AllGroups: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/AllGroups')

Assertion: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Assertion')

Asset: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Asset')

AssetCollection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/AssetCollection')

AssetScope: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/AssetScope')

ConflictTerm: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/ConflictTerm')

Constraint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Constraint')

Duty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Duty')

Group: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Group')

Individual: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Individual')

LeftOperand: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/LeftOperand')

LogicalConstraint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/LogicalConstraint')

Offer: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Offer')

Operator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Operator')

Party: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Party')

PartyCollection: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/PartyCollection')

PartyScope: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/PartyScope')

Permission: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Permission')

Policy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Policy')

Privacy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Privacy')

Prohibition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Prohibition')

Request: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Request')

RightOperand: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/RightOperand')

Rule: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Rule')

Set: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Set')

Ticket: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/Ticket')

UndefinedTerm: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/UndefinedTerm')

absolutePosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/absolutePosition')

absoluteSize: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/absoluteSize')

absoluteSpatialPosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/absoluteSpatialPosition')

absoluteTemporalPosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/absoluteTemporalPosition')

acceptTracking: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/acceptTracking')

action: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/action')

adHocShare: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/adHocShare')

aggregate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/aggregate')

andSequence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/andSequence')

annotate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/annotate')

anonymize: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/anonymize')

append: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/append')

appendTo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/appendTo')

archive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/archive')

assignee: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/assignee')

assigneeOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/assigneeOf')

assigner: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/assigner')

assignerOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/assignerOf')

attachPolicy: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/attachPolicy')

attachSource: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/attachSource')

attribute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/attribute')

attributedParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/attributedParty')

attributingParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/attributingParty')

commercialize: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/commercialize')

compensate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/compensate')

compensatedParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/compensatedParty')

compensatingParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/compensatingParty')

concurrentUse: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/concurrentUse')

conflict: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/conflict')

consentedParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/consentedParty')

consentingParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/consentingParty')

consequence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/consequence')

constraint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/constraint')

contractedParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/contractedParty')

contractingParty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/contractingParty')

copy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/copy')

core: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/core')

count: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/count')

dataType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/dataType')

dateTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/dateTime')

delayPeriod: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/delayPeriod')

delete: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/delete')

deliveryChannel: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/deliveryChannel')

derive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/derive')

device: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/device')

digitize: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/digitize')

display: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/display')

distribute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/distribute')

duty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/duty')

elapsedTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/elapsedTime')

ensureExclusivity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/ensureExclusivity')

eq: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/eq')

event: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/event')

execute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/execute')

export: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/export')

extract: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/extract')

extractChar: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/extractChar')

extractPage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/extractPage')

extractWord: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/extractWord')

failure: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/failure')

fileFormat: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/fileFormat')

function: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/function')

give: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/give')

grantUse: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/grantUse')

gt: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/gt')

gteq: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/gteq')

hasPart: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/hasPart')

hasPolicy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/hasPolicy')

ignore: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/ignore')

implies: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/implies')

include: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/include')

includedIn: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/includedIn')

index: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/index')

industry: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/industry')

inform: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/inform')

informedParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/informedParty')

informingParty: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/informingParty')

inheritAllowed: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/inheritAllowed')

inheritFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/inheritFrom')

inheritRelation: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/inheritRelation')

install: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/install')

invalid: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/invalid')

isA: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/isA')

isAllOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/isAllOf')

isAnyOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/isAnyOf')

isNoneOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/isNoneOf')

isPartOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/isPartOf')

language: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/language')

lease: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/lease')

leftOperand: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/leftOperand')

lend: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/lend')

license: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/license')

lt: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/lt')

lteq: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/lteq')

media: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/media')

meteredTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/meteredTime')

modify: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/modify')

move: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/move')

neq: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/neq')

nextPolicy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/nextPolicy')

obligation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/obligation')

obtainConsent: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/obtainConsent')

operand: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/operand')

operator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/operator')

output: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/output')

partOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/partOf')

pay: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/pay')

payAmount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/payAmount')

payeeParty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/payeeParty')

percentage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/percentage')

perm: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/perm')

permission: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/permission')

play: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/play')

policyUsage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/policyUsage')

present: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/present')

preview: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/preview')

print: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/print')

product: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/product')

profile: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/profile')

prohibit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/prohibit')

prohibition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/prohibition')

proximity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/proximity')

purpose: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/purpose')

read: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/read')

recipient: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/recipient')

refinement: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/refinement')

relation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/relation')

relativePosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/relativePosition')

relativeSize: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/relativeSize')

relativeSpatialPosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/relativeSpatialPosition')

relativeTemporalPosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/relativeTemporalPosition')

remedy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/remedy')

reproduce: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/reproduce')

resolution: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/resolution')

reviewPolicy: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/reviewPolicy')

rightOperand: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/rightOperand')

rightOperandReference: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/rightOperandReference')

scope: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/scope')

secondaryUse: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/secondaryUse')

sell: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/sell')

share: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/share')

shareAlike: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/shareAlike')

source: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/source')

spatial: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/spatial')

spatialCoordinates: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/spatialCoordinates')

status: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/status')

stream: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/stream')

support: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/support')

synchronize: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/synchronize')

system: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/system')

systemDevice: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/systemDevice')

target: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/target')

textToSpeech: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/textToSpeech')

timeInterval: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/timeInterval')

timedCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/timedCount')

trackedParty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/trackedParty')

trackingParty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/trackingParty')

transfer: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/transfer')

transform: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/transform')

translate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/translate')

uid: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/uid')

undefined: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/undefined')

uninstall: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/uninstall')

unit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/unit')

unitOfCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/unitOfCount')

use: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/use')

version: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/version')

virtualLocation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/virtualLocation')

watermark: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/watermark')

write: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/write')

writeTo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/writeTo')

xone: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/odrl/2/xone')

class rdflib.ORG

Bases: *DefinedNamespace*

Core organization ontology

Vocabulary for describing organizational structures, specializable to a broad variety of types of organization.

Generated from: http://www.w3.org/ns/org# Date: 2020-05-26 14:20:02.908408

ChangeEvent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#ChangeEvent')

FormalOrganization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#FormalOrganization')

Head: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#Head')

Membership: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#Membership')

Organization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#Organization')

OrganizationalCollaboration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#OrganizationalCollaboration')

OrganizationalUnit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#OrganizationalUnit')

Post: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#Post')

Role: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#Role')

Site: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#Site')

basedAt: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#basedAt')

changedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#changedBy')

classification: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#classification')

hasMember: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasMember')

hasMembership: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasMembership')

hasPost: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasPost')

hasPrimarySite: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasPrimarySite')

hasRegisteredSite: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasRegisteredSite')

hasSite: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasSite')

hasSubOrganization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasSubOrganization')

hasUnit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#hasUnit')

headOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#headOf')

heldBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#heldBy')

holds: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#holds')

identifier: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#identifier')

linkedTo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#linkedTo')

location: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#location')

member: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#member')

memberDuring: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#memberDuring')

memberOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#memberOf')

organization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#organization')

originalOrganization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#originalOrganization')

postIn: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#postIn')

purpose: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#purpose')

remuneration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#remuneration')

reportsTo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#reportsTo')

resultedFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#resultedFrom')

resultingOrganization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#resultingOrganization')

role: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#role')

roleProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#roleProperty')

siteAddress: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#siteAddress')

siteOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#siteOf')

subOrganizationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#subOrganizationOf')

transitiveSubOrganizationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#transitiveSubOrganizationOf')

unitOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/org#unitOf')

class rdflib.**OWL**

Bases: *DefinedNamespace*

The OWL 2 Schema vocabulary (OWL 2)

This ontology partially describes the built-in classes and properties that together form the basis of the RDF/XML syntax of OWL 2. The content of this ontology is based on Tables 6.1 and 6.2 in Section 6.4 of the OWL 2 RDF-Based Semantics specification, available at http://www.w3.org/TR/owl2-rdf-based- semantics/. Please note that those tables do not include the different annotations (labels, comments and rdfs:isDefinedBy links) used in this file. Also note that the descriptions provided in this ontology do not provide a complete and correct formal

description of either the syntax or the semantics of the introduced terms (please see the OWL 2 recommendations for the complete and normative specifications). Furthermore, the information provided by this ontology may be misleading if not used with care. This ontology SHOULD NOT be imported into OWL ontologies. Importing this file into an OWL 2 DL ontology will cause it to become an OWL 2 Full ontology and may have other, unexpected, consequences.

Generated from: http://www.w3.org/2002/07/owl# Date: 2020-05-26 14:20:03.193795

AllDifferent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#AllDifferent')

AllDisjointClasses: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#AllDisjointClasses')

AllDisjointProperties: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#AllDisjointProperties')

Annotation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Annotation')

AnnotationProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#AnnotationProperty')

AsymmetricProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#AsymmetricProperty')

Axiom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Axiom')

Class: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Class')

DataRange: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#DataRange')

DatatypeProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#DatatypeProperty')

DeprecatedClass: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#DeprecatedClass')

DeprecatedProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#DeprecatedProperty')

FunctionalProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#FunctionalProperty')

InverseFunctionalProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#InverseFunctionalProperty')

IrreflexiveProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#IrreflexiveProperty')

NamedIndividual: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#NamedIndividual')

NegativePropertyAssertion: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#NegativePropertyAssertion')

Nothing: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Nothing')

ObjectProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#ObjectProperty')

Ontology: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Ontology')

OntologyProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#OntologyProperty')

ReflexiveProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#ReflexiveProperty')

Restriction: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Restriction')

SymmetricProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#SymmetricProperty')

Thing: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Thing')

TransitiveProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#TransitiveProperty')

allValuesFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#allValuesFrom')

annotatedProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#annotatedProperty')

annotatedSource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#annotatedSource')

annotatedTarget: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#annotatedTarget')

assertionProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#assertionProperty')

backwardCompatibleWith: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#backwardCompatibleWith')

bottomDataProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#bottomDataProperty')

bottomObjectProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#bottomObjectProperty')

cardinality: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#cardinality')

complementOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#complementOf')

datatypeComplementOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#datatypeComplementOf')

deprecated: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#deprecated')

differentFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#differentFrom')

disjointUnionOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#disjointUnionOf')

disjointWith: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#disjointWith')

distinctMembers: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#distinctMembers')

equivalentClass: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#equivalentClass')

equivalentProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#equivalentProperty')

hasKey: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#hasKey')

hasSelf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#hasSelf')

hasValue: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#hasValue')

imports: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#imports')

incompatibleWith: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#incompatibleWith')

intersectionOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#intersectionOf')

inverseOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#inverseOf')

maxCardinality: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#maxCardinality')

maxQualifiedCardinality: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#maxQualifiedCardinality')

members: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#members')

minCardinality: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#minCardinality')

minQualifiedCardinality: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#minQualifiedCardinality')

onClass: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#onClass')

onDataRange: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#onDataRange')

onDatatype: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#onDatatype')

onProperties: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#onProperties')

onProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#onProperty')

oneOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#oneOf')

priorVersion: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#priorVersion')

propertyChainAxiom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#propertyChainAxiom')

propertyDisjointWith: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#propertyDisjointWith')

qualifiedCardinality: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#qualifiedCardinality')

rational: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#rational')

real: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#real')

sameAs: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#sameAs')

someValuesFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#someValuesFrom')

sourceIndividual: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#sourceIndividual')

targetIndividual: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#targetIndividual')

targetValue: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#targetValue')

topDataProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#topDataProperty')

topObjectProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#topObjectProperty')

unionOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#unionOf')

versionIRI: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#versionIRI')

versionInfo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#versionInfo')

withRestrictions: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2002/07/owl#withRestrictions')

class rdflib.**PROF**

Bases: *DefinedNamespace*

Profiles Vocabulary

This vocabulary is for describing relationships between standards/specifications, profiles of them and supporting artifacts such as validating resources. This model starts with [http://dublincore.org/2012/06/14/dcterms#Standard{]}(dct:Standard) entities which can either be Base Specifications (a standard not profiling any other Standard) or Profiles (Standards which do profile others). Base Specifications or Profiles can have Resource

Descriptors associated with them that defines implementing rules for the it. Resource Descriptors must indicate the role they play (to guide, to validate etc.) and the formalism they adhere to (dct:format) to allow for content negotiation. A vocabulary of Resource Roles are provided alongside this vocabulary but that list is extensible.

Generated from: https://www.w3.org/ns/dx/prof/profilesont.ttl Date: 2020-05-26 14:20:03.542924

Profile: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/Profile')**

ResourceDescriptor: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/ResourceDescriptor')**

ResourceRole: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/ResourceRole')**

hasArtifact: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/hasArtifact')**

hasResource: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/hasResource')**

hasRole: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/hasRole')**

hasToken: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/hasToken')**

isInheritedFrom: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/isInheritedFrom')**

isProfileOf: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/isProfileOf')**

isTransitiveProfileOf: *URIRef* = **rdflib.term.URIRef('http://www.w3.org/ns/dx/prof/isTransitiveProfileOf')**

class rdflib.**PROV**

Bases: *DefinedNamespace*

W3C PROVenance Interchange Ontology (PROV-O)

This document is published by the Provenance Working Group (http://www.w3.org/2011/prov/wiki/Main_Page). If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe public-prov-comments-request@w3.org, archives http://lists.w3.org/Archives/Public/public-prov-comments/). All feedback is welcome.

PROV Access and Query Ontology

This document is published by the Provenance Working Group (http://www.w3.org/2011/prov/wiki/Main_Page). If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe public-prov-comments-request@w3.org, archives http://lists.w3.org/Archives/Public/public-prov-comments/). All feedback is welcome.

Dublin Core extensions of the W3C PROVenance Interchange Ontology (PROV-O)

This document is published by the Provenance Working Group (http://www.w3.org/2011/prov/wiki/Main_Page). If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe public-prov-comments-request@w3.org, archives http://lists.w3.org/Archives/Public/public-prov-comments/). All feedback is welcome.

W3C PROV Linking Across Provenance Bundles Ontology (PROV-LINKS)

This document is published by the Provenance Working Group (http://www.w3.org/2011/prov/wiki/Main_Page). If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe public-prov-comments-request@w3.org, archives http://lists.w3.org/Archives/Public/public-prov-comments/). All feedback is welcome.

W3C PROVenance Interchange Ontology (PROV-O) Dictionary Extension

This document is published by the Provenance Working Group (http://www.w3.org/2011/prov/wiki/Main_Page). If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe public-prov-comments-request@w3.org, archives http://lists.w3.org/Archives/Public/public-prov-comments/). All feedback is welcome.

W3C PROVenance Interchange

This document is published by the Provenance Working Group (http://www.w3.org/2011/prov/wiki/Main_Page). If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe public-prov-comments-request@w3.org, archives http://lists.w3.org/ Archives/Public/public-prov-comments/). All feedback is welcome.

Generated from: http://www.w3.org/ns/prov Date: 2020-05-26 14:20:04.650279

Accept: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Accept')

Activity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Activity')

ActivityInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#ActivityInfluence')

Agent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Agent')

AgentInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#AgentInfluence')

Association: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Association')

Attribution: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Attribution')

Bundle: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Bundle')

Collection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Collection')

Communication: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Communication')

Contribute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Contribute')

Contributor: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Contributor')

Copyright: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Copyright')

Create: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Create')

Creator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Creator')

Delegation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Delegation')

Derivation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Derivation')

Dictionary: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Dictionary')

DirectQueryService: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#DirectQueryService')

EmptyCollection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#EmptyCollection')

EmptyDictionary: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#EmptyDictionary')

End: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#End')

Entity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Entity')

EntityInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#EntityInfluence')

Generation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Generation')

Influence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Influence')

Insertion: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Insertion')

InstantaneousEvent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#InstantaneousEvent')

Invalidation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Invalidation')

KeyEntityPair: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#KeyEntityPair')

Location: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Location')

Modify: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Modify')

Organization: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Organization')

Person: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Person')

Plan: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Plan')

PrimarySource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#PrimarySource')

Publish: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Publish')

Publisher: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Publisher')

Quotation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Quotation')

Removal: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Removal')

Replace: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Replace')

Revision: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Revision')

RightsAssignment: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#RightsAssignment')

RightsHolder: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#RightsHolder')

Role: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Role')

ServiceDescription: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#ServiceDescription')

SoftwareAgent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#SoftwareAgent')

Start: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Start')

Submit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Submit')

Usage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#Usage')

actedOnBehalfOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#actedOnBehalfOf')

activity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#activity')

activityOfInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#activityOfInfluence')

agent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#agent')

agentOfInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#agentOfInfluence')

alternateOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#alternateOf')

aq: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#aq')

asInBundle: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#asInBundle')

atLocation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#atLocation')

atTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#atTime')

category: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#category')

component: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#component')

constraints: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#constraints')

contributed: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#contributed')

definition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#definition')

derivedByInsertionFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#derivedByInsertionFrom')

derivedByRemovalFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#derivedByRemovalFrom')

describesService: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#describesService')

dictionary: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#dictionary')

dm: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#dm')

editorialNote: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#editorialNote')

editorsDefinition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#editorsDefinition')

ended: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#ended')

endedAtTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#endedAtTime')

entity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#entity')

entityOfInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#entityOfInfluence')

generalizationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#generalizationOf')

generated: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#generated')

generatedAsDerivation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#generatedAsDerivation')

generatedAtTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#generatedAtTime')

hadActivity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadActivity')

hadDelegate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadDelegate')

hadDerivation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadDerivation')

hadDictionaryMember: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadDictionaryMember')

hadGeneration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadGeneration')

hadInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadInfluence')

hadMember: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadMember')

hadPlan: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadPlan')

hadPrimarySource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadPrimarySource')

hadRevision: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadRevision')

hadRole: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadRole')

hadUsage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadUsage')

has_anchor: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#has_anchor')

has_provenance: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#has_provenance')

has_query_service: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#has_query_service')

influenced: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#influenced')

influencer: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#influencer')

informed: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#informed')

insertedKeyEntityPair: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#insertedKeyEntityPair')

invalidated: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#invalidated')

invalidatedAtTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#invalidatedAtTime')

inverse: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#inverse')

locationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#locationOf')

mentionOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#mentionOf')

n: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#n')

order: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#order')

pairEntity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#pairEntity')

pairKey: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#pairKey')

pingback: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#pingback')

provenanceUriTemplate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#provenanceUriTemplate')

qualifiedAssociation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedAssociation')

qualifiedAssociationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedAssociationOf')

qualifiedAttribution: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedAttribution')

qualifiedAttributionOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedAttributionOf')

qualifiedCommunication: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedCommunication')

qualifiedCommunicationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedCommunicationOf')

qualifiedDelegation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedDelegation')

qualifiedDelegationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedDelegationOf')

qualifiedDerivation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedDerivation')

qualifiedDerivationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedDerivationOf')

qualifiedEnd: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedEnd')

qualifiedEndOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedEndOf')

qualifiedForm: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedForm')

qualifiedGeneration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedGeneration')

qualifiedGenerationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedGenerationOf')

qualifiedInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedInfluence')

qualifiedInfluenceOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedInfluenceOf')

qualifiedInsertion: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedInsertion')

qualifiedInvalidation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedInvalidation')

qualifiedInvalidationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedInvalidationOf')

qualifiedPrimarySource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedPrimarySource')

qualifiedQuotation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedQuotation')

qualifiedQuotationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedQuotationOf')

qualifiedRemoval: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedRemoval')

qualifiedRevision: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedRevision')

qualifiedSourceOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedSourceOf')

qualifiedStart: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedStart')

qualifiedStartOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedStartOf')

qualifiedUsage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedUsage')

qualifiedUsingActivity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#qualifiedUsingActivity')

quotedAs: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#quotedAs')

removedKey: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#removedKey')

revisedEntity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#revisedEntity')

sharesDefinitionWith: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#sharesDefinitionWith')

specializationOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#specializationOf')

started: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#started')

startedAtTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#startedAtTime')

todo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#todo')

unqualifiedForm: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#unqualifiedForm')

used: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#used')

value: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#value')

wasActivityOfInfluence: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasActivityOfInfluence')

wasAssociateFor: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasAssociateFor')

wasAssociatedWith: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasAssociatedWith')

wasAttributedTo: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasAttributedTo')

wasDerivedFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasDerivedFrom')

wasEndedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasEndedBy')

wasGeneratedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasGeneratedBy')

wasInfluencedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasInfluencedBy')

wasInformedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasInformedBy')

wasInvalidatedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasInvalidatedBy')

wasMemberOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasMemberOf')

wasPlanOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasPlanOf')

wasPrimarySourceOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasPrimarySourceOf')

wasQuotedFrom: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasQuotedFrom')

wasRevisionOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasRevisionOf')

wasRoleIn: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasRoleIn')

wasStartedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasStartedBy')

wasUsedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasUsedBy')

wasUsedInDerivation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasUsedInDerivation')

class rdflib.QB

Bases: *DefinedNamespace*

Vocabulary for multi-dimensional (e.g. statistical) data publishing

This vocabulary allows multi-dimensional data, such as statistics, to be published in RDF. It is based on the core information model from SDMX (and thus also DDI).

Generated from: http://purl.org/linked-data/cube# Date: 2020-05-26 14:20:05.485176

Attachable: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#Attachable')

AttributeProperty: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#AttributeProperty')

CodedProperty: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#CodedProperty')

ComponentProperty: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#ComponentProperty')

ComponentSet: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#ComponentSet')

ComponentSpecification: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#ComponentSpecification')

DataSet: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#DataSet')

DataStructureDefinition: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#DataStructureDefinition')

DimensionProperty: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#DimensionProperty')

HierarchicalCodeList: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#HierarchicalCodeList')

MeasureProperty: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#MeasureProperty')

Observation: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#Observation')

ObservationGroup: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#ObservationGroup')

Slice: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#Slice')

SliceKey: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#SliceKey')

attribute: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#attribute')

codeList: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#codeList')

component: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#component')

componentAttachment: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#componentAttachment')

componentProperty: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#componentProperty')

componentRequired: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#componentRequired')

concept: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#concept')

dataSet: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#dataSet')

dimension: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#dimension')

hierarchyRoot: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#hierarchyRoot')

measure: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#measure')

measureDimension: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#measureDimension')

measureType: *URIRef* =
rdflib.term.URIRef('http://purl.org/linked-data/cube#measureType')

observation: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#observation')

observationGroup: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#observationGroup')

order: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#order')

parentChildProperty: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#parentChildProperty')

slice: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#slice')

sliceKey: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#sliceKey')

sliceStructure: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#sliceStructure')

structure: *URIRef* = rdflib.term.URIRef('http://purl.org/linked-data/cube#structure')

class rdflib.**RDF**

Bases: *DefinedNamespace*

The RDF Concepts Vocabulary (RDF)

This is the RDF Schema for the RDF vocabulary terms in the RDF Namespace, defined in RDF 1.1 Concepts.

Generated from: http://www.w3.org/1999/02/22-rdf-syntax-ns# Date: 2020-05-26 14:20:05.642859

dc:date "2019-12-16"

Alt: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt')

Bag: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag')

CompoundLiteral: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#CompoundLiteral')

HTML: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#HTML')

JSON: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#JSON')

List: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#List')

PlainLiteral: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral')

Property: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#Property')

Seq: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq')

Statement: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement')

XMLLiteral: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral')

direction: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#direction')

first: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#first')

langString: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#langString')

language: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#language')

nil: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#nil')

object: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#object')

predicate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate')

rest: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#rest')

subject: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#subject')

type: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type')

value: *URIRef* = rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#value')

class rdflib.**RDFS**

Bases: *DefinedNamespace*

The RDF Schema vocabulary (RDFS)

Generated from: http://www.w3.org/2000/01/rdf-schema# Date: 2020-05-26 14:20:05.794866

Class: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Class')

Container: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Container')

ContainerMembershipProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#ContainerMembershipProperty')

Datatype: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Datatype')

Literal: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Literal')

Resource: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#Resource')

comment: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#comment')

domain: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#domain')

isDefinedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#isDefinedBy')

label: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#label')

member: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#member')

range: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#range')

seeAlso: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#seeAlso')

subClassOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#subClassOf')

subPropertyOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2000/01/rdf-schema#subPropertyOf')

class rdflib.SDO

Bases: *DefinedNamespace*

schema.org namespace elements

3DModel, True, False & yield are not available as they collde with Python terms

Generated from: https://schema.org/version/latest/schemaorg-current-https.jsonld Date: 2021-12-01 By: Nicholas J. Car

AMRadioChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/AMRadioChannel')

APIReference: *URIRef* = rdflib.term.URIRef('https://schema.org/APIReference')

Abdomen: *URIRef* = rdflib.term.URIRef('https://schema.org/Abdomen')

AboutPage: *URIRef* = rdflib.term.URIRef('https://schema.org/AboutPage')

AcceptAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AcceptAction')

Accommodation: *URIRef* = rdflib.term.URIRef('https://schema.org/Accommodation')

AccountingService: *URIRef* = rdflib.term.URIRef('https://schema.org/AccountingService')

AchieveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AchieveAction')

Action: *URIRef* = rdflib.term.URIRef('https://schema.org/Action')

ActionAccessSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/ActionAccessSpecification')

ActionStatusType: *URIRef* = rdflib.term.URIRef('https://schema.org/ActionStatusType')

ActivateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ActivateAction')

ActivationFee: *URIRef* = rdflib.term.URIRef('https://schema.org/ActivationFee')

ActiveActionStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/ActiveActionStatus')

ActiveNotRecruiting: *URIRef* = rdflib.term.URIRef('https://schema.org/ActiveNotRecruiting')

AddAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AddAction')

AdministrativeArea: *URIRef* = rdflib.term.URIRef('https://schema.org/AdministrativeArea')

AdultEntertainment: *URIRef* = rdflib.term.URIRef('https://schema.org/AdultEntertainment')

AdvertiserContentArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/AdvertiserContentArticle')

AerobicActivity: *URIRef* = rdflib.term.URIRef('https://schema.org/AerobicActivity')

AggregateOffer: *URIRef* = rdflib.term.URIRef('https://schema.org/AggregateOffer')

AggregateRating: *URIRef* = rdflib.term.URIRef('https://schema.org/AggregateRating')

AgreeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AgreeAction')

Airline: *URIRef* = rdflib.term.URIRef('https://schema.org/Airline')

Airport: *URIRef* = rdflib.term.URIRef('https://schema.org/Airport')

AlbumRelease: *URIRef* = rdflib.term.URIRef('https://schema.org/AlbumRelease')

AlignmentObject: *URIRef* = rdflib.term.URIRef('https://schema.org/AlignmentObject')

AllWheelDriveConfiguration: *URIRef* = rdflib.term.URIRef('https://schema.org/AllWheelDriveConfiguration')

AllergiesHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/AllergiesHealthAspect')

AllocateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AllocateAction')

AmpStory: *URIRef* = rdflib.term.URIRef('https://schema.org/AmpStory')

AmusementPark: *URIRef* = rdflib.term.URIRef('https://schema.org/AmusementPark')

AnaerobicActivity: *URIRef* = rdflib.term.URIRef('https://schema.org/AnaerobicActivity')

AnalysisNewsArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/AnalysisNewsArticle')

AnatomicalStructure: *URIRef* = rdflib.term.URIRef('https://schema.org/AnatomicalStructure')

AnatomicalSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/AnatomicalSystem')

Anesthesia: *URIRef* = rdflib.term.URIRef('https://schema.org/Anesthesia')

AnimalShelter: *URIRef* = rdflib.term.URIRef('https://schema.org/AnimalShelter')

Answer: *URIRef* = rdflib.term.URIRef('https://schema.org/Answer')

Apartment: *URIRef* = rdflib.term.URIRef('https://schema.org/Apartment')

ApartmentComplex: *URIRef* =
rdflib.term.URIRef('https://schema.org/ApartmentComplex')

Appearance: *URIRef* = rdflib.term.URIRef('https://schema.org/Appearance')

AppendAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AppendAction')

ApplyAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ApplyAction')

ApprovedIndication: *URIRef* =
rdflib.term.URIRef('https://schema.org/ApprovedIndication')

Aquarium: *URIRef* = rdflib.term.URIRef('https://schema.org/Aquarium')

ArchiveComponent: *URIRef* =
rdflib.term.URIRef('https://schema.org/ArchiveComponent')

ArchiveOrganization: *URIRef* =
rdflib.term.URIRef('https://schema.org/ArchiveOrganization')

ArriveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ArriveAction')

ArtGallery: *URIRef* = rdflib.term.URIRef('https://schema.org/ArtGallery')

Artery: *URIRef* = rdflib.term.URIRef('https://schema.org/Artery')

Article: *URIRef* = rdflib.term.URIRef('https://schema.org/Article')

AskAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AskAction')

AskPublicNewsArticle: *URIRef* =
rdflib.term.URIRef('https://schema.org/AskPublicNewsArticle')

AssessAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AssessAction')

AssignAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AssignAction')

Atlas: *URIRef* = rdflib.term.URIRef('https://schema.org/Atlas')

Attorney: *URIRef* = rdflib.term.URIRef('https://schema.org/Attorney')

Audience: *URIRef* = rdflib.term.URIRef('https://schema.org/Audience')

AudioObject: *URIRef* = rdflib.term.URIRef('https://schema.org/AudioObject')

AudioObjectSnapshot: *URIRef* =
rdflib.term.URIRef('https://schema.org/AudioObjectSnapshot')

Audiobook: *URIRef* = rdflib.term.URIRef('https://schema.org/Audiobook')

AudiobookFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/AudiobookFormat')

AuthoritativeLegalValue: *URIRef* = rdflib.term.URIRef('https://schema.org/AuthoritativeLegalValue')

AuthorizeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/AuthorizeAction')

AutoBodyShop: *URIRef* = rdflib.term.URIRef('https://schema.org/AutoBodyShop')

AutoDealer: *URIRef* = rdflib.term.URIRef('https://schema.org/AutoDealer')

AutoPartsStore: *URIRef* = rdflib.term.URIRef('https://schema.org/AutoPartsStore')

AutoRental: *URIRef* = rdflib.term.URIRef('https://schema.org/AutoRental')

AutoRepair: *URIRef* = rdflib.term.URIRef('https://schema.org/AutoRepair')

AutoWash: *URIRef* = rdflib.term.URIRef('https://schema.org/AutoWash')

AutomatedTeller: *URIRef* = rdflib.term.URIRef('https://schema.org/AutomatedTeller')

AutomotiveBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/AutomotiveBusiness')

Ayurvedic: *URIRef* = rdflib.term.URIRef('https://schema.org/Ayurvedic')

BackOrder: *URIRef* = rdflib.term.URIRef('https://schema.org/BackOrder')

BackgroundNewsArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/BackgroundNewsArticle')

Bacteria: *URIRef* = rdflib.term.URIRef('https://schema.org/Bacteria')

Bakery: *URIRef* = rdflib.term.URIRef('https://schema.org/Bakery')

Balance: *URIRef* = rdflib.term.URIRef('https://schema.org/Balance')

BankAccount: *URIRef* = rdflib.term.URIRef('https://schema.org/BankAccount')

BankOrCreditUnion: *URIRef* = rdflib.term.URIRef('https://schema.org/BankOrCreditUnion')

BarOrPub: *URIRef* = rdflib.term.URIRef('https://schema.org/BarOrPub')

Barcode: *URIRef* = rdflib.term.URIRef('https://schema.org/Barcode')

BasicIncome: *URIRef* = rdflib.term.URIRef('https://schema.org/BasicIncome')

Beach: *URIRef* = rdflib.term.URIRef('https://schema.org/Beach')

BeautySalon: *URIRef* = rdflib.term.URIRef('https://schema.org/BeautySalon')

BedAndBreakfast: *URIRef* = rdflib.term.URIRef('https://schema.org/BedAndBreakfast')

BedDetails: *URIRef* = rdflib.term.URIRef('https://schema.org/BedDetails')

BedType: *URIRef* = rdflib.term.URIRef('https://schema.org/BedType')

BefriendAction: *URIRef* = rdflib.term.URIRef('https://schema.org/BefriendAction')

BenefitsHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/BenefitsHealthAspect')

BikeStore: *URIRef* = rdflib.term.URIRef('https://schema.org/BikeStore')

BioChemEntity: *URIRef* = rdflib.term.URIRef('https://schema.org/BioChemEntity')

Blog: *URIRef* = rdflib.term.URIRef('https://schema.org/Blog')

BlogPosting: *URIRef* = rdflib.term.URIRef('https://schema.org/BlogPosting')

BloodTest: *URIRef* = rdflib.term.URIRef('https://schema.org/BloodTest')

BoardingPolicyType: *URIRef* =
rdflib.term.URIRef('https://schema.org/BoardingPolicyType')

BoatReservation: *URIRef* = rdflib.term.URIRef('https://schema.org/BoatReservation')

BoatTerminal: *URIRef* = rdflib.term.URIRef('https://schema.org/BoatTerminal')

BoatTrip: *URIRef* = rdflib.term.URIRef('https://schema.org/BoatTrip')

BodyMeasurementArm: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementArm')

BodyMeasurementBust: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementBust')

BodyMeasurementChest: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementChest')

BodyMeasurementFoot: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementFoot')

BodyMeasurementHand: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementHand')

BodyMeasurementHead: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementHead')

BodyMeasurementHeight: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementHeight')

BodyMeasurementHips: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementHips')

BodyMeasurementInsideLeg: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementInsideLeg')

BodyMeasurementNeck: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementNeck')

BodyMeasurementTypeEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementTypeEnumeration')

BodyMeasurementUnderbust: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementUnderbust')

BodyMeasurementWaist: *URIRef* =
rdflib.term.URIRef('https://schema.org/BodyMeasurementWaist')

BodyMeasurementWeight: *URIRef* = rdflib.term.URIRef('https://schema.org/BodyMeasurementWeight')

BodyOfWater: *URIRef* = rdflib.term.URIRef('https://schema.org/BodyOfWater')

Bone: *URIRef* = rdflib.term.URIRef('https://schema.org/Bone')

Book: *URIRef* = rdflib.term.URIRef('https://schema.org/Book')

BookFormatType: *URIRef* = rdflib.term.URIRef('https://schema.org/BookFormatType')

BookSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/BookSeries')

BookStore: *URIRef* = rdflib.term.URIRef('https://schema.org/BookStore')

BookmarkAction: *URIRef* = rdflib.term.URIRef('https://schema.org/BookmarkAction')

Boolean: *URIRef* = rdflib.term.URIRef('https://schema.org/Boolean')

BorrowAction: *URIRef* = rdflib.term.URIRef('https://schema.org/BorrowAction')

BowlingAlley: *URIRef* = rdflib.term.URIRef('https://schema.org/BowlingAlley')

BrainStructure: *URIRef* = rdflib.term.URIRef('https://schema.org/BrainStructure')

Brand: *URIRef* = rdflib.term.URIRef('https://schema.org/Brand')

BreadcrumbList: *URIRef* = rdflib.term.URIRef('https://schema.org/BreadcrumbList')

Brewery: *URIRef* = rdflib.term.URIRef('https://schema.org/Brewery')

Bridge: *URIRef* = rdflib.term.URIRef('https://schema.org/Bridge')

BroadcastChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/BroadcastChannel')

BroadcastEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/BroadcastEvent')

BroadcastFrequencySpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/BroadcastFrequencySpecification')

BroadcastRelease: *URIRef* = rdflib.term.URIRef('https://schema.org/BroadcastRelease')

BroadcastService: *URIRef* = rdflib.term.URIRef('https://schema.org/BroadcastService')

BrokerageAccount: *URIRef* = rdflib.term.URIRef('https://schema.org/BrokerageAccount')

BuddhistTemple: *URIRef* = rdflib.term.URIRef('https://schema.org/BuddhistTemple')

BusOrCoach: *URIRef* = rdflib.term.URIRef('https://schema.org/BusOrCoach')

BusReservation: *URIRef* = rdflib.term.URIRef('https://schema.org/BusReservation')

BusStation: *URIRef* = rdflib.term.URIRef('https://schema.org/BusStation')

BusStop: *URIRef* = rdflib.term.URIRef('https://schema.org/BusStop')

BusTrip: *URIRef* = rdflib.term.URIRef('https://schema.org/BusTrip')

BusinessAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/BusinessAudience')

BusinessEntityType: *URIRef* = rdflib.term.URIRef('https://schema.org/BusinessEntityType')

BusinessEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/BusinessEvent')

BusinessFunction: *URIRef* = rdflib.term.URIRef('https://schema.org/BusinessFunction')

BusinessSupport: *URIRef* = rdflib.term.URIRef('https://schema.org/BusinessSupport')

BuyAction: *URIRef* = rdflib.term.URIRef('https://schema.org/BuyAction')

CDCPMDRecord: *URIRef* = rdflib.term.URIRef('https://schema.org/CDCPMDRecord')

CDFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/CDFormat')

CT: *URIRef* = rdflib.term.URIRef('https://schema.org/CT')

CableOrSatelliteService: *URIRef* = rdflib.term.URIRef('https://schema.org/CableOrSatelliteService')

CafeOrCoffeeShop: *URIRef* = rdflib.term.URIRef('https://schema.org/CafeOrCoffeeShop')

Campground: *URIRef* = rdflib.term.URIRef('https://schema.org/Campground')

CampingPitch: *URIRef* = rdflib.term.URIRef('https://schema.org/CampingPitch')

Canal: *URIRef* = rdflib.term.URIRef('https://schema.org/Canal')

CancelAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CancelAction')

Car: *URIRef* = rdflib.term.URIRef('https://schema.org/Car')

CarUsageType: *URIRef* = rdflib.term.URIRef('https://schema.org/CarUsageType')

Cardiovascular: *URIRef* = rdflib.term.URIRef('https://schema.org/Cardiovascular')

CardiovascularExam: *URIRef* = rdflib.term.URIRef('https://schema.org/CardiovascularExam')

CaseSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/CaseSeries')

Casino: *URIRef* = rdflib.term.URIRef('https://schema.org/Casino')

CassetteFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/CassetteFormat')

CategoryCode: *URIRef* = rdflib.term.URIRef('https://schema.org/CategoryCode')

CategoryCodeSet: *URIRef* = rdflib.term.URIRef('https://schema.org/CategoryCodeSet')

CatholicChurch: *URIRef* = rdflib.term.URIRef('https://schema.org/CatholicChurch')

CausesHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/CausesHealthAspect')

Cemetery: *URIRef* = rdflib.term.URIRef('https://schema.org/Cemetery')

Chapter: *URIRef* = rdflib.term.URIRef('https://schema.org/Chapter')

CharitableIncorporatedOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/CharitableIncorporatedOrganization')

CheckAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CheckAction')

CheckInAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CheckInAction')

CheckOutAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CheckOutAction')

CheckoutPage: *URIRef* = rdflib.term.URIRef('https://schema.org/CheckoutPage')

ChemicalSubstance: *URIRef* = rdflib.term.URIRef('https://schema.org/ChemicalSubstance')

ChildCare: *URIRef* = rdflib.term.URIRef('https://schema.org/ChildCare')

ChildrensEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/ChildrensEvent')

Chiropractic: *URIRef* = rdflib.term.URIRef('https://schema.org/Chiropractic')

ChooseAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ChooseAction')

Church: *URIRef* = rdflib.term.URIRef('https://schema.org/Church')

City: *URIRef* = rdflib.term.URIRef('https://schema.org/City')

CityHall: *URIRef* = rdflib.term.URIRef('https://schema.org/CityHall')

CivicStructure: *URIRef* = rdflib.term.URIRef('https://schema.org/CivicStructure')

Claim: *URIRef* = rdflib.term.URIRef('https://schema.org/Claim')

ClaimReview: *URIRef* = rdflib.term.URIRef('https://schema.org/ClaimReview')

Class: *URIRef* = rdflib.term.URIRef('https://schema.org/Class')

CleaningFee: *URIRef* = rdflib.term.URIRef('https://schema.org/CleaningFee')

Clinician: *URIRef* = rdflib.term.URIRef('https://schema.org/Clinician')

Clip: *URIRef* = rdflib.term.URIRef('https://schema.org/Clip')

ClothingStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ClothingStore')

CoOp: *URIRef* = rdflib.term.URIRef('https://schema.org/CoOp')

Code: *URIRef* = rdflib.term.URIRef('https://schema.org/Code')

CohortStudy: *URIRef* = rdflib.term.URIRef('https://schema.org/CohortStudy')

Collection: *URIRef* = rdflib.term.URIRef('https://schema.org/Collection')

CollectionPage: *URIRef* = rdflib.term.URIRef('https://schema.org/CollectionPage')

CollegeOrUniversity: *URIRef* = rdflib.term.URIRef('https://schema.org/CollegeOrUniversity')

ComedyClub: *URIRef* = rdflib.term.URIRef('https://schema.org/ComedyClub')

ComedyEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/ComedyEvent')

ComicCoverArt: *URIRef* = rdflib.term.URIRef('https://schema.org/ComicCoverArt')

ComicIssue: *URIRef* = rdflib.term.URIRef('https://schema.org/ComicIssue')

ComicSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/ComicSeries')

ComicStory: *URIRef* = rdflib.term.URIRef('https://schema.org/ComicStory')

Comment: *URIRef* = rdflib.term.URIRef('https://schema.org/Comment')

CommentAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CommentAction')

CommentPermission: *URIRef* = rdflib.term.URIRef('https://schema.org/CommentPermission')

CommunicateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CommunicateAction')

CommunityHealth: *URIRef* = rdflib.term.URIRef('https://schema.org/CommunityHealth')

CompilationAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/CompilationAlbum')

CompleteDataFeed: *URIRef* = rdflib.term.URIRef('https://schema.org/CompleteDataFeed')

Completed: *URIRef* = rdflib.term.URIRef('https://schema.org/Completed')

CompletedActionStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/CompletedActionStatus')

CompoundPriceSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/CompoundPriceSpecification')

ComputerLanguage: *URIRef* = rdflib.term.URIRef('https://schema.org/ComputerLanguage')

ComputerStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ComputerStore')

ConfirmAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ConfirmAction')

Consortium: *URIRef* = rdflib.term.URIRef('https://schema.org/Consortium')

ConsumeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ConsumeAction')

ContactPage: *URIRef* = rdflib.term.URIRef('https://schema.org/ContactPage')

ContactPoint: *URIRef* = rdflib.term.URIRef('https://schema.org/ContactPoint')

ContactPointOption: *URIRef* = rdflib.term.URIRef('https://schema.org/ContactPointOption')

ContagiousnessHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/ContagiousnessHealthAspect')

Continent: *URIRef* = rdflib.term.URIRef('https://schema.org/Continent')

ControlAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ControlAction')

ConvenienceStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ConvenienceStore')

Conversation: *URIRef* = rdflib.term.URIRef('https://schema.org/Conversation')

CookAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CookAction')

Corporation: *URIRef* = rdflib.term.URIRef('https://schema.org/Corporation')

CorrectionComment: *URIRef* = rdflib.term.URIRef('https://schema.org/CorrectionComment')

Country: *URIRef* = rdflib.term.URIRef('https://schema.org/Country')

Course: *URIRef* = rdflib.term.URIRef('https://schema.org/Course')

CourseInstance: *URIRef* = rdflib.term.URIRef('https://schema.org/CourseInstance')

Courthouse: *URIRef* = rdflib.term.URIRef('https://schema.org/Courthouse')

CoverArt: *URIRef* = rdflib.term.URIRef('https://schema.org/CoverArt')

CovidTestingFacility: *URIRef* = rdflib.term.URIRef('https://schema.org/CovidTestingFacility')

CreateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/CreateAction')

CreativeWork: *URIRef* = rdflib.term.URIRef('https://schema.org/CreativeWork')

CreativeWorkSeason: *URIRef* = rdflib.term.URIRef('https://schema.org/CreativeWorkSeason')

CreativeWorkSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/CreativeWorkSeries')

CreditCard: *URIRef* = rdflib.term.URIRef('https://schema.org/CreditCard')

Crematorium: *URIRef* = rdflib.term.URIRef('https://schema.org/Crematorium')

CriticReview: *URIRef* = rdflib.term.URIRef('https://schema.org/CriticReview')

CrossSectional: *URIRef* = rdflib.term.URIRef('https://schema.org/CrossSectional')

CssSelectorType: *URIRef* = rdflib.term.URIRef('https://schema.org/CssSelectorType')

CurrencyConversionService: *URIRef* = rdflib.term.URIRef('https://schema.org/CurrencyConversionService')

DDxElement: *URIRef* = rdflib.term.URIRef('https://schema.org/DDxElement')

DJMixAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/DJMixAlbum')

DVDFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/DVDFormat')

DamagedCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/DamagedCondition')

DanceEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/DanceEvent')

DanceGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/DanceGroup')

DataCatalog: *URIRef* = rdflib.term.URIRef('https://schema.org/DataCatalog')

DataDownload: *URIRef* = rdflib.term.URIRef('https://schema.org/DataDownload')

DataFeed: *URIRef* = rdflib.term.URIRef('https://schema.org/DataFeed')

DataFeedItem: *URIRef* = rdflib.term.URIRef('https://schema.org/DataFeedItem')

DataType: *URIRef* = rdflib.term.URIRef('https://schema.org/DataType')

Dataset: *URIRef* = rdflib.term.URIRef('https://schema.org/Dataset')

Date: *URIRef* = rdflib.term.URIRef('https://schema.org/Date')

DateTime: *URIRef* = rdflib.term.URIRef('https://schema.org/DateTime')

DatedMoneySpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/DatedMoneySpecification')

DayOfWeek: *URIRef* = rdflib.term.URIRef('https://schema.org/DayOfWeek')

DaySpa: *URIRef* = rdflib.term.URIRef('https://schema.org/DaySpa')

DeactivateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DeactivateAction')

DecontextualizedContent: *URIRef* = rdflib.term.URIRef('https://schema.org/DecontextualizedContent')

DefenceEstablishment: *URIRef* = rdflib.term.URIRef('https://schema.org/DefenceEstablishment')

DefinedRegion: *URIRef* = rdflib.term.URIRef('https://schema.org/DefinedRegion')

DefinedTerm: *URIRef* = rdflib.term.URIRef('https://schema.org/DefinedTerm')

DefinedTermSet: *URIRef* = rdflib.term.URIRef('https://schema.org/DefinedTermSet')

DefinitiveLegalValue: *URIRef* = rdflib.term.URIRef('https://schema.org/DefinitiveLegalValue')

DeleteAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DeleteAction')

DeliveryChargeSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/DeliveryChargeSpecification')

DeliveryEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/DeliveryEvent')

DeliveryMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/DeliveryMethod')

DeliveryTimeSettings: *URIRef* = rdflib.term.URIRef('https://schema.org/DeliveryTimeSettings')

Demand: *URIRef* = rdflib.term.URIRef('https://schema.org/Demand')

DemoAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/DemoAlbum')

Dentist: *URIRef* = rdflib.term.URIRef('https://schema.org/Dentist')

Dentistry: *URIRef* = rdflib.term.URIRef('https://schema.org/Dentistry')

DepartAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DepartAction')

DepartmentStore: *URIRef* = rdflib.term.URIRef('https://schema.org/DepartmentStore')

DepositAccount: *URIRef* = rdflib.term.URIRef('https://schema.org/DepositAccount')

Dermatologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Dermatologic')

Dermatology: *URIRef* = rdflib.term.URIRef('https://schema.org/Dermatology')

DiabeticDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/DiabeticDiet')

Diagnostic: *URIRef* = rdflib.term.URIRef('https://schema.org/Diagnostic')

DiagnosticLab: *URIRef* = rdflib.term.URIRef('https://schema.org/DiagnosticLab')

DiagnosticProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/DiagnosticProcedure')

Diet: *URIRef* = rdflib.term.URIRef('https://schema.org/Diet')

DietNutrition: *URIRef* = rdflib.term.URIRef('https://schema.org/DietNutrition')

DietarySupplement: *URIRef* = rdflib.term.URIRef('https://schema.org/DietarySupplement')

DigitalAudioTapeFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/DigitalAudioTapeFormat')

DigitalDocument: *URIRef* = rdflib.term.URIRef('https://schema.org/DigitalDocument')

DigitalDocumentPermission: *URIRef* = rdflib.term.URIRef('https://schema.org/DigitalDocumentPermission')

DigitalDocumentPermissionType: *URIRef* = rdflib.term.URIRef('https://schema.org/DigitalDocumentPermissionType')

DigitalFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/DigitalFormat')

DisabilitySupport: *URIRef* = rdflib.term.URIRef('https://schema.org/DisabilitySupport')

DisagreeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DisagreeAction')

Discontinued: *URIRef* = rdflib.term.URIRef('https://schema.org/Discontinued')

DiscoverAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DiscoverAction')

DiscussionForumPosting: *URIRef* = rdflib.term.URIRef('https://schema.org/DiscussionForumPosting')

DislikeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DislikeAction')

Distance: *URIRef* = rdflib.term.URIRef('https://schema.org/Distance')

DistanceFee: *URIRef* = rdflib.term.URIRef('https://schema.org/DistanceFee')

Distillery: *URIRef* = rdflib.term.URIRef('https://schema.org/Distillery')

DonateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DonateAction')

DoseSchedule: *URIRef* = rdflib.term.URIRef('https://schema.org/DoseSchedule')

DoubleBlindedTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/DoubleBlindedTrial')

DownloadAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DownloadAction')

Downpayment: *URIRef* = rdflib.term.URIRef('https://schema.org/Downpayment')

DrawAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DrawAction')

Drawing: *URIRef* = rdflib.term.URIRef('https://schema.org/Drawing')

DrinkAction: *URIRef* = rdflib.term.URIRef('https://schema.org/DrinkAction')

DriveWheelConfigurationValue: *URIRef* = rdflib.term.URIRef('https://schema.org/DriveWheelConfigurationValue')

DrivingSchoolVehicleUsage: *URIRef* = rdflib.term.URIRef('https://schema.org/DrivingSchoolVehicleUsage')

Drug: *URIRef* = rdflib.term.URIRef('https://schema.org/Drug')

DrugClass: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugClass')

DrugCost: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugCost')

DrugCostCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugCostCategory')

DrugLegalStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugLegalStatus')

DrugPregnancyCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugPregnancyCategory')

DrugPrescriptionStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugPrescriptionStatus')

DrugStrength: *URIRef* = rdflib.term.URIRef('https://schema.org/DrugStrength')

DryCleaningOrLaundry: *URIRef* = rdflib.term.URIRef('https://schema.org/DryCleaningOrLaundry')

Duration: *URIRef* = rdflib.term.URIRef('https://schema.org/Duration')

EBook: *URIRef* = rdflib.term.URIRef('https://schema.org/EBook')

EPRelease: *URIRef* = rdflib.term.URIRef('https://schema.org/EPRelease')

EUEnergyEfficiencyCategoryA: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryA')

EUEnergyEfficiencyCategoryA1Plus: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryA1Plus')

EUEnergyEfficiencyCategoryA2Plus: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryA2Plus')

EUEnergyEfficiencyCategoryA3Plus: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryA3Plus')

EUEnergyEfficiencyCategoryB: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryB')

EUEnergyEfficiencyCategoryC: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryC')

EUEnergyEfficiencyCategoryD: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryD')

EUEnergyEfficiencyCategoryE: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryE')

EUEnergyEfficiencyCategoryF: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryF')

EUEnergyEfficiencyCategoryG: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyCategoryG')

EUEnergyEfficiencyEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/EUEnergyEfficiencyEnumeration')

Ear: *URIRef* = rdflib.term.URIRef('https://schema.org/Ear')

EatAction: *URIRef* = rdflib.term.URIRef('https://schema.org/EatAction')

EditedOrCroppedContent: *URIRef* =
rdflib.term.URIRef('https://schema.org/EditedOrCroppedContent')

EducationEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/EducationEvent')

EducationalAudience: *URIRef* =
rdflib.term.URIRef('https://schema.org/EducationalAudience')

EducationalOccupationalCredential: *URIRef* =
rdflib.term.URIRef('https://schema.org/EducationalOccupationalCredential')

EducationalOccupationalProgram: *URIRef* =
rdflib.term.URIRef('https://schema.org/EducationalOccupationalProgram')

EducationalOrganization: *URIRef* =
rdflib.term.URIRef('https://schema.org/EducationalOrganization')

EffectivenessHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/EffectivenessHealthAspect')

Electrician: *URIRef* = rdflib.term.URIRef('https://schema.org/Electrician')

ElectronicsStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ElectronicsStore')

ElementarySchool: *URIRef* = rdflib.term.URIRef('https://schema.org/ElementarySchool')

EmailMessage: *URIRef* = rdflib.term.URIRef('https://schema.org/EmailMessage')

Embassy: *URIRef* = rdflib.term.URIRef('https://schema.org/Embassy')

Emergency: *URIRef* = rdflib.term.URIRef('https://schema.org/Emergency')

EmergencyService: *URIRef* = rdflib.term.URIRef('https://schema.org/EmergencyService')

EmployeeRole: *URIRef* = rdflib.term.URIRef('https://schema.org/EmployeeRole')

EmployerAggregateRating: *URIRef* = rdflib.term.URIRef('https://schema.org/EmployerAggregateRating')

EmployerReview: *URIRef* = rdflib.term.URIRef('https://schema.org/EmployerReview')

EmploymentAgency: *URIRef* = rdflib.term.URIRef('https://schema.org/EmploymentAgency')

Endocrine: *URIRef* = rdflib.term.URIRef('https://schema.org/Endocrine')

EndorseAction: *URIRef* = rdflib.term.URIRef('https://schema.org/EndorseAction')

EndorsementRating: *URIRef* = rdflib.term.URIRef('https://schema.org/EndorsementRating')

Energy: *URIRef* = rdflib.term.URIRef('https://schema.org/Energy')

EnergyConsumptionDetails: *URIRef* = rdflib.term.URIRef('https://schema.org/EnergyConsumptionDetails')

EnergyEfficiencyEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/EnergyEfficiencyEnumeration')

EnergyStarCertified: *URIRef* = rdflib.term.URIRef('https://schema.org/EnergyStarCertified')

EnergyStarEnergyEfficiencyEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/EnergyStarEnergyEfficiencyEnumeration')

EngineSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/EngineSpecification')

EnrollingByInvitation: *URIRef* = rdflib.term.URIRef('https://schema.org/EnrollingByInvitation')

EntertainmentBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/EntertainmentBusiness')

EntryPoint: *URIRef* = rdflib.term.URIRef('https://schema.org/EntryPoint')

Enumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/Enumeration')

Episode: *URIRef* = rdflib.term.URIRef('https://schema.org/Episode')

Event: *URIRef* = rdflib.term.URIRef('https://schema.org/Event')

EventAttendanceModeEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/EventAttendanceModeEnumeration')

EventCancelled: *URIRef* = rdflib.term.URIRef('https://schema.org/EventCancelled')

EventMovedOnline: *URIRef* = rdflib.term.URIRef('https://schema.org/EventMovedOnline')

EventPostponed: *URIRef* = rdflib.term.URIRef('https://schema.org/EventPostponed')

EventRescheduled: *URIRef* = rdflib.term.URIRef('https://schema.org/EventRescheduled')

EventReservation: *URIRef* = rdflib.term.URIRef('https://schema.org/EventReservation')

EventScheduled: *URIRef* = rdflib.term.URIRef('https://schema.org/EventScheduled')

EventSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/EventSeries')

EventStatusType: *URIRef* = rdflib.term.URIRef('https://schema.org/EventStatusType')

EventVenue: *URIRef* = rdflib.term.URIRef('https://schema.org/EventVenue')

EvidenceLevelA: *URIRef* = rdflib.term.URIRef('https://schema.org/EvidenceLevelA')

EvidenceLevelB: *URIRef* = rdflib.term.URIRef('https://schema.org/EvidenceLevelB')

EvidenceLevelC: *URIRef* = rdflib.term.URIRef('https://schema.org/EvidenceLevelC')

ExchangeRateSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/ExchangeRateSpecification')

ExchangeRefund: *URIRef* = rdflib.term.URIRef('https://schema.org/ExchangeRefund')

ExerciseAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ExerciseAction')

ExerciseGym: *URIRef* = rdflib.term.URIRef('https://schema.org/ExerciseGym')

ExercisePlan: *URIRef* = rdflib.term.URIRef('https://schema.org/ExercisePlan')

ExhibitionEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/ExhibitionEvent')

Eye: *URIRef* = rdflib.term.URIRef('https://schema.org/Eye')

FAQPage: *URIRef* = rdflib.term.URIRef('https://schema.org/FAQPage')

FDAcategoryA: *URIRef* = rdflib.term.URIRef('https://schema.org/FDAcategoryA')

FDAcategoryB: *URIRef* = rdflib.term.URIRef('https://schema.org/FDAcategoryB')

FDAcategoryC: *URIRef* = rdflib.term.URIRef('https://schema.org/FDAcategoryC')

FDAcategoryD: *URIRef* = rdflib.term.URIRef('https://schema.org/FDAcategoryD')

FDAcategoryX: *URIRef* = rdflib.term.URIRef('https://schema.org/FDAcategoryX')

FDAnotEvaluated: *URIRef* = rdflib.term.URIRef('https://schema.org/FDAnotEvaluated')

FMRadioChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/FMRadioChannel')

FailedActionStatus: *URIRef* =
rdflib.term.URIRef('https://schema.org/FailedActionStatus')

FastFoodRestaurant: *URIRef* =
rdflib.term.URIRef('https://schema.org/FastFoodRestaurant')

Female: *URIRef* = rdflib.term.URIRef('https://schema.org/Female')

Festival: *URIRef* = rdflib.term.URIRef('https://schema.org/Festival')

FilmAction: *URIRef* = rdflib.term.URIRef('https://schema.org/FilmAction')

FinancialProduct: *URIRef* =
rdflib.term.URIRef('https://schema.org/FinancialProduct')

FinancialService: *URIRef* =
rdflib.term.URIRef('https://schema.org/FinancialService')

FindAction: *URIRef* = rdflib.term.URIRef('https://schema.org/FindAction')

FireStation: *URIRef* = rdflib.term.URIRef('https://schema.org/FireStation')

Flexibility: *URIRef* = rdflib.term.URIRef('https://schema.org/Flexibility')

Flight: *URIRef* = rdflib.term.URIRef('https://schema.org/Flight')

FlightReservation: *URIRef* =
rdflib.term.URIRef('https://schema.org/FlightReservation')

Float: *URIRef* = rdflib.term.URIRef('https://schema.org/Float')

FloorPlan: *URIRef* = rdflib.term.URIRef('https://schema.org/FloorPlan')

Florist: *URIRef* = rdflib.term.URIRef('https://schema.org/Florist')

FollowAction: *URIRef* = rdflib.term.URIRef('https://schema.org/FollowAction')

FoodEstablishment: *URIRef* =
rdflib.term.URIRef('https://schema.org/FoodEstablishment')

FoodEstablishmentReservation: *URIRef* =
rdflib.term.URIRef('https://schema.org/FoodEstablishmentReservation')

FoodEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/FoodEvent')

FoodService: *URIRef* = rdflib.term.URIRef('https://schema.org/FoodService')

FourWheelDriveConfiguration: *URIRef* = rdflib.term.URIRef('https://schema.org/FourWheelDriveConfiguration')

FreeReturn: *URIRef* = rdflib.term.URIRef('https://schema.org/FreeReturn')

Friday: *URIRef* = rdflib.term.URIRef('https://schema.org/Friday')

FrontWheelDriveConfiguration: *URIRef* = rdflib.term.URIRef('https://schema.org/FrontWheelDriveConfiguration')

FullRefund: *URIRef* = rdflib.term.URIRef('https://schema.org/FullRefund')

FundingAgency: *URIRef* = rdflib.term.URIRef('https://schema.org/FundingAgency')

FundingScheme: *URIRef* = rdflib.term.URIRef('https://schema.org/FundingScheme')

Fungus: *URIRef* = rdflib.term.URIRef('https://schema.org/Fungus')

FurnitureStore: *URIRef* = rdflib.term.URIRef('https://schema.org/FurnitureStore')

Game: *URIRef* = rdflib.term.URIRef('https://schema.org/Game')

GamePlayMode: *URIRef* = rdflib.term.URIRef('https://schema.org/GamePlayMode')

GameServer: *URIRef* = rdflib.term.URIRef('https://schema.org/GameServer')

GameServerStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/GameServerStatus')

GardenStore: *URIRef* = rdflib.term.URIRef('https://schema.org/GardenStore')

GasStation: *URIRef* = rdflib.term.URIRef('https://schema.org/GasStation')

Gastroenterologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Gastroenterologic')

GatedResidenceCommunity: *URIRef* = rdflib.term.URIRef('https://schema.org/GatedResidenceCommunity')

GenderType: *URIRef* = rdflib.term.URIRef('https://schema.org/GenderType')

Gene: *URIRef* = rdflib.term.URIRef('https://schema.org/Gene')

GeneralContractor: *URIRef* = rdflib.term.URIRef('https://schema.org/GeneralContractor')

Genetic: *URIRef* = rdflib.term.URIRef('https://schema.org/Genetic')

Genitourinary: *URIRef* = rdflib.term.URIRef('https://schema.org/Genitourinary')

GeoCircle: *URIRef* = rdflib.term.URIRef('https://schema.org/GeoCircle')

GeoCoordinates: *URIRef* = rdflib.term.URIRef('https://schema.org/GeoCoordinates')

GeoShape: *URIRef* = rdflib.term.URIRef('https://schema.org/GeoShape')

GeospatialGeometry: *URIRef* = rdflib.term.URIRef('https://schema.org/GeospatialGeometry')

Geriatric: *URIRef* = rdflib.term.URIRef('https://schema.org/Geriatric')

GettingAccessHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/GettingAccessHealthAspect')

GiveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/GiveAction')

GlutenFreeDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/GlutenFreeDiet')

GolfCourse: *URIRef* = rdflib.term.URIRef('https://schema.org/GolfCourse')

GovernmentBenefitsType: *URIRef* = rdflib.term.URIRef('https://schema.org/GovernmentBenefitsType')

GovernmentBuilding: *URIRef* = rdflib.term.URIRef('https://schema.org/GovernmentBuilding')

GovernmentOffice: *URIRef* = rdflib.term.URIRef('https://schema.org/GovernmentOffice')

GovernmentOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/GovernmentOrganization')

GovernmentPermit: *URIRef* = rdflib.term.URIRef('https://schema.org/GovernmentPermit')

GovernmentService: *URIRef* = rdflib.term.URIRef('https://schema.org/GovernmentService')

Grant: *URIRef* = rdflib.term.URIRef('https://schema.org/Grant')

GraphicNovel: *URIRef* = rdflib.term.URIRef('https://schema.org/GraphicNovel')

GroceryStore: *URIRef* = rdflib.term.URIRef('https://schema.org/GroceryStore')

GroupBoardingPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/GroupBoardingPolicy')

Guide: *URIRef* = rdflib.term.URIRef('https://schema.org/Guide')

Gynecologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Gynecologic')

HVACBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/HVACBusiness')

Hackathon: *URIRef* = rdflib.term.URIRef('https://schema.org/Hackathon')

HairSalon: *URIRef* = rdflib.term.URIRef('https://schema.org/HairSalon')

HalalDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/HalalDiet')

Hardcover: *URIRef* = rdflib.term.URIRef('https://schema.org/Hardcover')

HardwareStore: *URIRef* = rdflib.term.URIRef('https://schema.org/HardwareStore')

Head: *URIRef* = rdflib.term.URIRef('https://schema.org/Head')

HealthAndBeautyBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthAndBeautyBusiness')

HealthAspectEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthAspectEnumeration')

HealthCare: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthCare')

HealthClub: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthClub')

HealthInsurancePlan: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthInsurancePlan')

HealthPlanCostSharingSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthPlanCostSharingSpecification')

HealthPlanFormulary: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthPlanFormulary')

HealthPlanNetwork: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthPlanNetwork')

HealthTopicContent: *URIRef* = rdflib.term.URIRef('https://schema.org/HealthTopicContent')

HearingImpairedSupported: *URIRef* = rdflib.term.URIRef('https://schema.org/HearingImpairedSupported')

Hematologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Hematologic')

HighSchool: *URIRef* = rdflib.term.URIRef('https://schema.org/HighSchool')

HinduDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/HinduDiet')

HinduTemple: *URIRef* = rdflib.term.URIRef('https://schema.org/HinduTemple')

HobbyShop: *URIRef* = rdflib.term.URIRef('https://schema.org/HobbyShop')

HomeAndConstructionBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/HomeAndConstructionBusiness')

HomeGoodsStore: *URIRef* = rdflib.term.URIRef('https://schema.org/HomeGoodsStore')

Homeopathic: *URIRef* = rdflib.term.URIRef('https://schema.org/Homeopathic')

Hospital: *URIRef* = rdflib.term.URIRef('https://schema.org/Hospital')

Hostel: *URIRef* = rdflib.term.URIRef('https://schema.org/Hostel')

Hotel: *URIRef* = rdflib.term.URIRef('https://schema.org/Hotel')

HotelRoom: *URIRef* = rdflib.term.URIRef('https://schema.org/HotelRoom')

House: *URIRef* = rdflib.term.URIRef('https://schema.org/House')

HousePainter: *URIRef* = rdflib.term.URIRef('https://schema.org/HousePainter')

HowItWorksHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/HowItWorksHealthAspect')

HowOrWhereHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/HowOrWhereHealthAspect')

HowTo: *URIRef* = rdflib.term.URIRef('https://schema.org/HowTo')

HowToDirection: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToDirection')

HowToItem: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToItem')

HowToSection: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToSection')

HowToStep: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToStep')

HowToSupply: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToSupply')

HowToTip: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToTip')

HowToTool: *URIRef* = rdflib.term.URIRef('https://schema.org/HowToTool')

HyperToc: *URIRef* = rdflib.term.URIRef('https://schema.org/HyperToc')

HyperTocEntry: *URIRef* = rdflib.term.URIRef('https://schema.org/HyperTocEntry')

IceCreamShop: *URIRef* = rdflib.term.URIRef('https://schema.org/IceCreamShop')

IgnoreAction: *URIRef* = rdflib.term.URIRef('https://schema.org/IgnoreAction')

ImageGallery: *URIRef* = rdflib.term.URIRef('https://schema.org/ImageGallery')

ImageObject: *URIRef* = rdflib.term.URIRef('https://schema.org/ImageObject')

ImageObjectSnapshot: *URIRef* =
rdflib.term.URIRef('https://schema.org/ImageObjectSnapshot')

ImagingTest: *URIRef* = rdflib.term.URIRef('https://schema.org/ImagingTest')

InForce: *URIRef* = rdflib.term.URIRef('https://schema.org/InForce')

InStock: *URIRef* = rdflib.term.URIRef('https://schema.org/InStock')

InStoreOnly: *URIRef* = rdflib.term.URIRef('https://schema.org/InStoreOnly')

IndividualProduct: *URIRef* =
rdflib.term.URIRef('https://schema.org/IndividualProduct')

Infectious: *URIRef* = rdflib.term.URIRef('https://schema.org/Infectious')

InfectiousAgentClass: *URIRef* =
rdflib.term.URIRef('https://schema.org/InfectiousAgentClass')

InfectiousDisease: *URIRef* =
rdflib.term.URIRef('https://schema.org/InfectiousDisease')

InformAction: *URIRef* = rdflib.term.URIRef('https://schema.org/InformAction')

IngredientsHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/IngredientsHealthAspect')

InsertAction: *URIRef* = rdflib.term.URIRef('https://schema.org/InsertAction')

InstallAction: *URIRef* = rdflib.term.URIRef('https://schema.org/InstallAction')

Installment: *URIRef* = rdflib.term.URIRef('https://schema.org/Installment')

InsuranceAgency: *URIRef* = rdflib.term.URIRef('https://schema.org/InsuranceAgency')

Intangible: *URIRef* = rdflib.term.URIRef('https://schema.org/Intangible')

Integer: *URIRef* = rdflib.term.URIRef('https://schema.org/Integer')

InteractAction: *URIRef* = rdflib.term.URIRef('https://schema.org/InteractAction')

InteractionCounter: *URIRef* =
rdflib.term.URIRef('https://schema.org/InteractionCounter')

InternationalTrial: *URIRef* =
rdflib.term.URIRef('https://schema.org/InternationalTrial')

InternetCafe: *URIRef* = rdflib.term.URIRef('https://schema.org/InternetCafe')

InvestmentFund: *URIRef* = rdflib.term.URIRef('https://schema.org/InvestmentFund')

InvestmentOrDeposit: *URIRef* =
rdflib.term.URIRef('https://schema.org/InvestmentOrDeposit')

InviteAction: *URIRef* = rdflib.term.URIRef('https://schema.org/InviteAction')

Invoice: *URIRef* = rdflib.term.URIRef('https://schema.org/Invoice')

InvoicePrice: *URIRef* = rdflib.term.URIRef('https://schema.org/InvoicePrice')

ItemAvailability: *URIRef* =
rdflib.term.URIRef('https://schema.org/ItemAvailability')

ItemList: *URIRef* = rdflib.term.URIRef('https://schema.org/ItemList')

ItemListOrderAscending: *URIRef* =
rdflib.term.URIRef('https://schema.org/ItemListOrderAscending')

ItemListOrderDescending: *URIRef* =
rdflib.term.URIRef('https://schema.org/ItemListOrderDescending')

ItemListOrderType: *URIRef* =
rdflib.term.URIRef('https://schema.org/ItemListOrderType')

ItemListUnordered: *URIRef* =
rdflib.term.URIRef('https://schema.org/ItemListUnordered')

ItemPage: *URIRef* = rdflib.term.URIRef('https://schema.org/ItemPage')

JewelryStore: *URIRef* = rdflib.term.URIRef('https://schema.org/JewelryStore')

JobPosting: *URIRef* = rdflib.term.URIRef('https://schema.org/JobPosting')

JoinAction: *URIRef* = rdflib.term.URIRef('https://schema.org/JoinAction')

Joint: *URIRef* = rdflib.term.URIRef('https://schema.org/Joint')

KosherDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/KosherDiet')

LaboratoryScience: *URIRef* =
rdflib.term.URIRef('https://schema.org/LaboratoryScience')

LakeBodyOfWater: *URIRef* = rdflib.term.URIRef('https://schema.org/LakeBodyOfWater')

Landform: *URIRef* = rdflib.term.URIRef('https://schema.org/Landform')

LandmarksOrHistoricalBuildings: *URIRef* = rdflib.term.URIRef('https://schema.org/LandmarksOrHistoricalBuildings')

Language: *URIRef* = rdflib.term.URIRef('https://schema.org/Language')

LaserDiscFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/LaserDiscFormat')

LearningResource: *URIRef* = rdflib.term.URIRef('https://schema.org/LearningResource')

LeaveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/LeaveAction')

LeftHandDriving: *URIRef* = rdflib.term.URIRef('https://schema.org/LeftHandDriving')

LegalForceStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/LegalForceStatus')

LegalService: *URIRef* = rdflib.term.URIRef('https://schema.org/LegalService')

LegalValueLevel: *URIRef* = rdflib.term.URIRef('https://schema.org/LegalValueLevel')

Legislation: *URIRef* = rdflib.term.URIRef('https://schema.org/Legislation')

LegislationObject: *URIRef* = rdflib.term.URIRef('https://schema.org/LegislationObject')

LegislativeBuilding: *URIRef* = rdflib.term.URIRef('https://schema.org/LegislativeBuilding')

LeisureTimeActivity: *URIRef* = rdflib.term.URIRef('https://schema.org/LeisureTimeActivity')

LendAction: *URIRef* = rdflib.term.URIRef('https://schema.org/LendAction')

Library: *URIRef* = rdflib.term.URIRef('https://schema.org/Library')

LibrarySystem: *URIRef* = rdflib.term.URIRef('https://schema.org/LibrarySystem')

LifestyleModification: *URIRef* = rdflib.term.URIRef('https://schema.org/LifestyleModification')

Ligament: *URIRef* = rdflib.term.URIRef('https://schema.org/Ligament')

LikeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/LikeAction')

LimitedAvailability: *URIRef* = rdflib.term.URIRef('https://schema.org/LimitedAvailability')

LimitedByGuaranteeCharity: *URIRef* = rdflib.term.URIRef('https://schema.org/LimitedByGuaranteeCharity')

LinkRole: *URIRef* = rdflib.term.URIRef('https://schema.org/LinkRole')

LiquorStore: *URIRef* = rdflib.term.URIRef('https://schema.org/LiquorStore')

ListItem: *URIRef* = rdflib.term.URIRef('https://schema.org/ListItem')

ListPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/ListPrice')

ListenAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ListenAction')

LiteraryEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/LiteraryEvent')

LiveAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/LiveAlbum')

LiveBlogPosting: *URIRef* = rdflib.term.URIRef('https://schema.org/LiveBlogPosting')

LivingWithHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/LivingWithHealthAspect')

LoanOrCredit: *URIRef* = rdflib.term.URIRef('https://schema.org/LoanOrCredit')

LocalBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/LocalBusiness')

LocationFeatureSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/LocationFeatureSpecification')

LockerDelivery: *URIRef* = rdflib.term.URIRef('https://schema.org/LockerDelivery')

Locksmith: *URIRef* = rdflib.term.URIRef('https://schema.org/Locksmith')

LodgingBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/LodgingBusiness')

LodgingReservation: *URIRef* = rdflib.term.URIRef('https://schema.org/LodgingReservation')

Longitudinal: *URIRef* = rdflib.term.URIRef('https://schema.org/Longitudinal')

LoseAction: *URIRef* = rdflib.term.URIRef('https://schema.org/LoseAction')

LowCalorieDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/LowCalorieDiet')

LowFatDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/LowFatDiet')

LowLactoseDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/LowLactoseDiet')

LowSaltDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/LowSaltDiet')

Lung: *URIRef* = rdflib.term.URIRef('https://schema.org/Lung')

LymphaticVessel: *URIRef* = rdflib.term.URIRef('https://schema.org/LymphaticVessel')

MRI: *URIRef* = rdflib.term.URIRef('https://schema.org/MRI')

MSRP: *URIRef* = rdflib.term.URIRef('https://schema.org/MSRP')

Male: *URIRef* = rdflib.term.URIRef('https://schema.org/Male')

Manuscript: *URIRef* = rdflib.term.URIRef('https://schema.org/Manuscript')

Map: *URIRef* = rdflib.term.URIRef('https://schema.org/Map')

MapCategoryType: *URIRef* = rdflib.term.URIRef('https://schema.org/MapCategoryType')

MarryAction: *URIRef* = rdflib.term.URIRef('https://schema.org/MarryAction')

Mass: *URIRef* = rdflib.term.URIRef('https://schema.org/Mass')

MathSolver: *URIRef* = rdflib.term.URIRef('https://schema.org/MathSolver')

MaximumDoseSchedule: *URIRef* =
rdflib.term.URIRef('https://schema.org/MaximumDoseSchedule')

MayTreatHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/MayTreatHealthAspect')

MeasurementTypeEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/MeasurementTypeEnumeration')

MediaGallery: *URIRef* = rdflib.term.URIRef('https://schema.org/MediaGallery')

MediaManipulationRatingEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/MediaManipulationRatingEnumeration')

MediaObject: *URIRef* = rdflib.term.URIRef('https://schema.org/MediaObject')

MediaReview: *URIRef* = rdflib.term.URIRef('https://schema.org/MediaReview')

MediaReviewItem: *URIRef* = rdflib.term.URIRef('https://schema.org/MediaReviewItem')

MediaSubscription: *URIRef* =
rdflib.term.URIRef('https://schema.org/MediaSubscription')

MedicalAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalAudience')

MedicalAudienceType: *URIRef* =
rdflib.term.URIRef('https://schema.org/MedicalAudienceType')

MedicalBusiness: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalBusiness')

MedicalCause: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalCause')

MedicalClinic: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalClinic')

MedicalCode: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalCode')

MedicalCondition: *URIRef* =
rdflib.term.URIRef('https://schema.org/MedicalCondition')

MedicalConditionStage: *URIRef* =
rdflib.term.URIRef('https://schema.org/MedicalConditionStage')

MedicalContraindication: *URIRef* =
rdflib.term.URIRef('https://schema.org/MedicalContraindication')

MedicalDevice: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalDevice')

MedicalDevicePurpose: *URIRef* =
rdflib.term.URIRef('https://schema.org/MedicalDevicePurpose')

MedicalEntity: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalEntity')

MedicalEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/MedicalEnumeration')

MedicalEvidenceLevel: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalEvidenceLevel')

MedicalGuideline: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalGuideline')

MedicalGuidelineContraindication: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalGuidelineContraindication')

MedicalGuidelineRecommendation: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalGuidelineRecommendation')

MedicalImagingTechnique: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalImagingTechnique')

MedicalIndication: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalIndication')

MedicalIntangible: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalIntangible')

MedicalObservationalStudy: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalObservationalStudy')

MedicalObservationalStudyDesign: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalObservationalStudyDesign')

MedicalOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalOrganization')

MedicalProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalProcedure')

MedicalProcedureType: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalProcedureType')

MedicalResearcher: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalResearcher')

MedicalRiskCalculator: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalRiskCalculator')

MedicalRiskEstimator: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalRiskEstimator')

MedicalRiskFactor: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalRiskFactor')

MedicalRiskScore: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalRiskScore')

MedicalScholarlyArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalScholarlyArticle')

MedicalSign: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalSign')

MedicalSignOrSymptom: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalSignOrSymptom')

MedicalSpecialty: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalSpecialty')

MedicalStudy: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalStudy')

MedicalStudyStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalStudyStatus')

MedicalSymptom: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalSymptom')

MedicalTest: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalTest')

MedicalTestPanel: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalTestPanel')

MedicalTherapy: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalTherapy')

MedicalTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalTrial')

MedicalTrialDesign: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalTrialDesign')

MedicalWebPage: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicalWebPage')

MedicineSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/MedicineSystem')

MeetingRoom: *URIRef* = rdflib.term.URIRef('https://schema.org/MeetingRoom')

MensClothingStore: *URIRef* = rdflib.term.URIRef('https://schema.org/MensClothingStore')

Menu: *URIRef* = rdflib.term.URIRef('https://schema.org/Menu')

MenuItem: *URIRef* = rdflib.term.URIRef('https://schema.org/MenuItem')

MenuSection: *URIRef* = rdflib.term.URIRef('https://schema.org/MenuSection')

MerchantReturnEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnEnumeration')

MerchantReturnFiniteReturnWindow: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnFiniteReturnWindow')

MerchantReturnNotPermitted: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnNotPermitted')

MerchantReturnPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnPolicy')

MerchantReturnPolicySeasonalOverride: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnPolicySeasonalOverride')

MerchantReturnUnlimitedWindow: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnUnlimitedWindow')

MerchantReturnUnspecified: *URIRef* = rdflib.term.URIRef('https://schema.org/MerchantReturnUnspecified')

Message: *URIRef* = rdflib.term.URIRef('https://schema.org/Message')

MiddleSchool: *URIRef* = rdflib.term.URIRef('https://schema.org/MiddleSchool')

Midwifery: *URIRef* = rdflib.term.URIRef('https://schema.org/Midwifery')

MinimumAdvertisedPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/MinimumAdvertisedPrice')

MisconceptionsHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/MisconceptionsHealthAspect')

MixedEventAttendanceMode: *URIRef* = rdflib.term.URIRef('https://schema.org/MixedEventAttendanceMode')

MixtapeAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/MixtapeAlbum')

MobileApplication: *URIRef* = rdflib.term.URIRef('https://schema.org/MobileApplication')

MobilePhoneStore: *URIRef* = rdflib.term.URIRef('https://schema.org/MobilePhoneStore')

MolecularEntity: *URIRef* = rdflib.term.URIRef('https://schema.org/MolecularEntity')

Monday: *URIRef* = rdflib.term.URIRef('https://schema.org/Monday')

MonetaryAmount: *URIRef* = rdflib.term.URIRef('https://schema.org/MonetaryAmount')

MonetaryAmountDistribution: *URIRef* = rdflib.term.URIRef('https://schema.org/MonetaryAmountDistribution')

MonetaryGrant: *URIRef* = rdflib.term.URIRef('https://schema.org/MonetaryGrant')

MoneyTransfer: *URIRef* = rdflib.term.URIRef('https://schema.org/MoneyTransfer')

MortgageLoan: *URIRef* = rdflib.term.URIRef('https://schema.org/MortgageLoan')

Mosque: *URIRef* = rdflib.term.URIRef('https://schema.org/Mosque')

Motel: *URIRef* = rdflib.term.URIRef('https://schema.org/Motel')

Motorcycle: *URIRef* = rdflib.term.URIRef('https://schema.org/Motorcycle')

MotorcycleDealer: *URIRef* = rdflib.term.URIRef('https://schema.org/MotorcycleDealer')

MotorcycleRepair: *URIRef* = rdflib.term.URIRef('https://schema.org/MotorcycleRepair')

MotorizedBicycle: *URIRef* = rdflib.term.URIRef('https://schema.org/MotorizedBicycle')

Mountain: *URIRef* = rdflib.term.URIRef('https://schema.org/Mountain')

MoveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/MoveAction')

Movie: *URIRef* = rdflib.term.URIRef('https://schema.org/Movie')

MovieClip: *URIRef* = rdflib.term.URIRef('https://schema.org/MovieClip')

MovieRentalStore: *URIRef* =
rdflib.term.URIRef('https://schema.org/MovieRentalStore')

MovieSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/MovieSeries')

MovieTheater: *URIRef* = rdflib.term.URIRef('https://schema.org/MovieTheater')

MovingCompany: *URIRef* = rdflib.term.URIRef('https://schema.org/MovingCompany')

MultiCenterTrial: *URIRef* =
rdflib.term.URIRef('https://schema.org/MultiCenterTrial')

MultiPlayer: *URIRef* = rdflib.term.URIRef('https://schema.org/MultiPlayer')

MulticellularParasite: *URIRef* =
rdflib.term.URIRef('https://schema.org/MulticellularParasite')

Muscle: *URIRef* = rdflib.term.URIRef('https://schema.org/Muscle')

Musculoskeletal: *URIRef* = rdflib.term.URIRef('https://schema.org/Musculoskeletal')

MusculoskeletalExam: *URIRef* =
rdflib.term.URIRef('https://schema.org/MusculoskeletalExam')

Museum: *URIRef* = rdflib.term.URIRef('https://schema.org/Museum')

MusicAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicAlbum')

MusicAlbumProductionType: *URIRef* =
rdflib.term.URIRef('https://schema.org/MusicAlbumProductionType')

MusicAlbumReleaseType: *URIRef* =
rdflib.term.URIRef('https://schema.org/MusicAlbumReleaseType')

MusicComposition: *URIRef* =
rdflib.term.URIRef('https://schema.org/MusicComposition')

MusicEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicEvent')

MusicGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicGroup')

MusicPlaylist: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicPlaylist')

MusicRecording: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicRecording')

MusicRelease: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicRelease')

MusicReleaseFormatType: *URIRef* =
rdflib.term.URIRef('https://schema.org/MusicReleaseFormatType')

MusicStore: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicStore')

MusicVenue: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicVenue')

MusicVideoObject: *URIRef* = rdflib.term.URIRef('https://schema.org/MusicVideoObject')

NGO: *URIRef* = rdflib.term.URIRef('https://schema.org/NGO')

NLNonprofitType: *URIRef* = rdflib.term.URIRef('https://schema.org/NLNonprofitType')

NailSalon: *URIRef* = rdflib.term.URIRef('https://schema.org/NailSalon')

Neck: *URIRef* = rdflib.term.URIRef('https://schema.org/Neck')

Nerve: *URIRef* = rdflib.term.URIRef('https://schema.org/Nerve')

Neuro: *URIRef* = rdflib.term.URIRef('https://schema.org/Neuro')

Neurologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Neurologic')

NewCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/NewCondition')

NewsArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/NewsArticle')

NewsMediaOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/NewsMediaOrganization')

Newspaper: *URIRef* = rdflib.term.URIRef('https://schema.org/Newspaper')

NightClub: *URIRef* = rdflib.term.URIRef('https://schema.org/NightClub')

NoninvasiveProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/NoninvasiveProcedure')

Nonprofit501a: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501a')

Nonprofit501c1: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c1')

Nonprofit501c10: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c10')

Nonprofit501c11: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c11')

Nonprofit501c12: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c12')

Nonprofit501c13: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c13')

Nonprofit501c14: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c14')

Nonprofit501c15: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c15')

Nonprofit501c16: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c16')

Nonprofit501c17: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c17')

Nonprofit501c18: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c18')

Nonprofit501c19: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c19')

Nonprofit501c2: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c2')

Nonprofit501c20: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c20')

Nonprofit501c21: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c21')

Nonprofit501c22: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c22')

Nonprofit501c23: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c23')

Nonprofit501c24: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c24')

Nonprofit501c25: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c25')

Nonprofit501c26: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c26')

Nonprofit501c27: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c27')

Nonprofit501c28: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c28')

Nonprofit501c3: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c3')

Nonprofit501c4: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c4')

Nonprofit501c5: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c5')

Nonprofit501c6: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c6')

Nonprofit501c7: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c7')

Nonprofit501c8: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c8')

Nonprofit501c9: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501c9')

Nonprofit501d: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501d')

Nonprofit501e: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501e')

Nonprofit501f: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501f')

Nonprofit501k: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501k')

Nonprofit501n: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501n')

Nonprofit501q: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit501q')

Nonprofit527: *URIRef* = rdflib.term.URIRef('https://schema.org/Nonprofit527')

NonprofitANBI: *URIRef* = rdflib.term.URIRef('https://schema.org/NonprofitANBI')

NonprofitSBBI: *URIRef* = rdflib.term.URIRef('https://schema.org/NonprofitSBBI')

NonprofitType: *URIRef* = rdflib.term.URIRef('https://schema.org/NonprofitType')

Nose: *URIRef* = rdflib.term.URIRef('https://schema.org/Nose')

NotInForce: *URIRef* = rdflib.term.URIRef('https://schema.org/NotInForce')

NotYetRecruiting: *URIRef* =
rdflib.term.URIRef('https://schema.org/NotYetRecruiting')

Notary: *URIRef* = rdflib.term.URIRef('https://schema.org/Notary')

NoteDigitalDocument: *URIRef* =
rdflib.term.URIRef('https://schema.org/NoteDigitalDocument')

Number: *URIRef* = rdflib.term.URIRef('https://schema.org/Number')

Nursing: *URIRef* = rdflib.term.URIRef('https://schema.org/Nursing')

NutritionInformation: *URIRef* =
rdflib.term.URIRef('https://schema.org/NutritionInformation')

OTC: *URIRef* = rdflib.term.URIRef('https://schema.org/OTC')

Observation: *URIRef* = rdflib.term.URIRef('https://schema.org/Observation')

Observational: *URIRef* = rdflib.term.URIRef('https://schema.org/Observational')

Obstetric: *URIRef* = rdflib.term.URIRef('https://schema.org/Obstetric')

Occupation: *URIRef* = rdflib.term.URIRef('https://schema.org/Occupation')

OccupationalActivity: *URIRef* =
rdflib.term.URIRef('https://schema.org/OccupationalActivity')

OccupationalExperienceRequirements: *URIRef* =
rdflib.term.URIRef('https://schema.org/OccupationalExperienceRequirements')

OccupationalTherapy: *URIRef* =
rdflib.term.URIRef('https://schema.org/OccupationalTherapy')

OceanBodyOfWater: *URIRef* =
rdflib.term.URIRef('https://schema.org/OceanBodyOfWater')

Offer: *URIRef* = rdflib.term.URIRef('https://schema.org/Offer')

OfferCatalog: *URIRef* = rdflib.term.URIRef('https://schema.org/OfferCatalog')

OfferForLease: *URIRef* = rdflib.term.URIRef('https://schema.org/OfferForLease')

OfferForPurchase: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfferForPurchase')

OfferItemCondition: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfferItemCondition')

OfferShippingDetails: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfferShippingDetails')

OfficeEquipmentStore: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfficeEquipmentStore')

OfficialLegalValue: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfficialLegalValue')

OfflineEventAttendanceMode: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfflineEventAttendanceMode')

OfflinePermanently: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfflinePermanently')

OfflineTemporarily: *URIRef* =
rdflib.term.URIRef('https://schema.org/OfflineTemporarily')

OnDemandEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/OnDemandEvent')

OnSitePickup: *URIRef* = rdflib.term.URIRef('https://schema.org/OnSitePickup')

Oncologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Oncologic')

OneTimePayments: *URIRef* = rdflib.term.URIRef('https://schema.org/OneTimePayments')

Online: *URIRef* = rdflib.term.URIRef('https://schema.org/Online')

OnlineEventAttendanceMode: *URIRef* = rdflib.term.URIRef('https://schema.org/OnlineEventAttendanceMode')

OnlineFull: *URIRef* = rdflib.term.URIRef('https://schema.org/OnlineFull')

OnlineOnly: *URIRef* = rdflib.term.URIRef('https://schema.org/OnlineOnly')

OpenTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/OpenTrial')

OpeningHoursSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/OpeningHoursSpecification')

OpinionNewsArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/OpinionNewsArticle')

Optician: *URIRef* = rdflib.term.URIRef('https://schema.org/Optician')

Optometric: *URIRef* = rdflib.term.URIRef('https://schema.org/Optometric')

Order: *URIRef* = rdflib.term.URIRef('https://schema.org/Order')

OrderAction: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderAction')

OrderCancelled: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderCancelled')

OrderDelivered: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderDelivered')

OrderInTransit: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderInTransit')

OrderItem: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderItem')

OrderPaymentDue: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderPaymentDue')

OrderPickupAvailable: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderPickupAvailable')

OrderProblem: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderProblem')

OrderProcessing: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderProcessing')

OrderReturned: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderReturned')

OrderStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/OrderStatus')

Organization: *URIRef* = rdflib.term.URIRef('https://schema.org/Organization')

OrganizationRole: *URIRef* = rdflib.term.URIRef('https://schema.org/OrganizationRole')

OrganizeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/OrganizeAction')

OriginalMediaContent: *URIRef* = rdflib.term.URIRef('https://schema.org/OriginalMediaContent')

OriginalShippingFees: *URIRef* = rdflib.term.URIRef('https://schema.org/OriginalShippingFees')

Osteopathic: *URIRef* = rdflib.term.URIRef('https://schema.org/Osteopathic')

Otolaryngologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Otolaryngologic')

OutOfStock: *URIRef* = rdflib.term.URIRef('https://schema.org/OutOfStock')

OutletStore: *URIRef* = rdflib.term.URIRef('https://schema.org/OutletStore')

OverviewHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/OverviewHealthAspect')

OwnershipInfo: *URIRef* = rdflib.term.URIRef('https://schema.org/OwnershipInfo')

PET: *URIRef* = rdflib.term.URIRef('https://schema.org/PET')

PaidLeave: *URIRef* = rdflib.term.URIRef('https://schema.org/PaidLeave')

PaintAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PaintAction')

Painting: *URIRef* = rdflib.term.URIRef('https://schema.org/Painting')

PalliativeProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/PalliativeProcedure')

Paperback: *URIRef* = rdflib.term.URIRef('https://schema.org/Paperback')

ParcelDelivery: *URIRef* = rdflib.term.URIRef('https://schema.org/ParcelDelivery')

ParcelService: *URIRef* = rdflib.term.URIRef('https://schema.org/ParcelService')

ParentAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/ParentAudience')

ParentalSupport: *URIRef* = rdflib.term.URIRef('https://schema.org/ParentalSupport')

Park: *URIRef* = rdflib.term.URIRef('https://schema.org/Park')

ParkingFacility: *URIRef* = rdflib.term.URIRef('https://schema.org/ParkingFacility')

ParkingMap: *URIRef* = rdflib.term.URIRef('https://schema.org/ParkingMap')

PartiallyInForce: *URIRef* = rdflib.term.URIRef('https://schema.org/PartiallyInForce')

Pathology: *URIRef* = rdflib.term.URIRef('https://schema.org/Pathology')

PathologyTest: *URIRef* = rdflib.term.URIRef('https://schema.org/PathologyTest')

Patient: *URIRef* = rdflib.term.URIRef('https://schema.org/Patient')

PatientExperienceHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/PatientExperienceHealthAspect')

PawnShop: *URIRef* = rdflib.term.URIRef('https://schema.org/PawnShop')

PayAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PayAction')

PaymentAutomaticallyApplied: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentAutomaticallyApplied')

PaymentCard: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentCard')

PaymentChargeSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentChargeSpecification')

PaymentComplete: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentComplete')

PaymentDeclined: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentDeclined')

PaymentDue: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentDue')

PaymentMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentMethod')

PaymentPastDue: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentPastDue')

PaymentService: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentService')

PaymentStatusType: *URIRef* = rdflib.term.URIRef('https://schema.org/PaymentStatusType')

Pediatric: *URIRef* = rdflib.term.URIRef('https://schema.org/Pediatric')

PeopleAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/PeopleAudience')

PercutaneousProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/PercutaneousProcedure')

PerformAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PerformAction')

PerformanceRole: *URIRef* = rdflib.term.URIRef('https://schema.org/PerformanceRole')

PerformingArtsTheater: *URIRef* = rdflib.term.URIRef('https://schema.org/PerformingArtsTheater')

PerformingGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/PerformingGroup')

Periodical: *URIRef* = rdflib.term.URIRef('https://schema.org/Periodical')

Permit: *URIRef* = rdflib.term.URIRef('https://schema.org/Permit')

Person: *URIRef* = rdflib.term.URIRef('https://schema.org/Person')

PetStore: *URIRef* = rdflib.term.URIRef('https://schema.org/PetStore')

Pharmacy: *URIRef* = rdflib.term.URIRef('https://schema.org/Pharmacy')

PharmacySpecialty: *URIRef* = rdflib.term.URIRef('https://schema.org/PharmacySpecialty')

Photograph: *URIRef* = rdflib.term.URIRef('https://schema.org/Photograph')

PhotographAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PhotographAction')

PhysicalActivity: *URIRef* = rdflib.term.URIRef('https://schema.org/PhysicalActivity')

PhysicalActivityCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/PhysicalActivityCategory')

PhysicalExam: *URIRef* = rdflib.term.URIRef('https://schema.org/PhysicalExam')

PhysicalTherapy: *URIRef* = rdflib.term.URIRef('https://schema.org/PhysicalTherapy')

Physician: *URIRef* = rdflib.term.URIRef('https://schema.org/Physician')

Physiotherapy: *URIRef* = rdflib.term.URIRef('https://schema.org/Physiotherapy')

Place: *URIRef* = rdflib.term.URIRef('https://schema.org/Place')

PlaceOfWorship: *URIRef* = rdflib.term.URIRef('https://schema.org/PlaceOfWorship')

PlaceboControlledTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/PlaceboControlledTrial')

PlanAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PlanAction')

PlasticSurgery: *URIRef* = rdflib.term.URIRef('https://schema.org/PlasticSurgery')

Play: *URIRef* = rdflib.term.URIRef('https://schema.org/Play')

PlayAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PlayAction')

Playground: *URIRef* = rdflib.term.URIRef('https://schema.org/Playground')

Plumber: *URIRef* = rdflib.term.URIRef('https://schema.org/Plumber')

PodcastEpisode: *URIRef* = rdflib.term.URIRef('https://schema.org/PodcastEpisode')

PodcastSeason: *URIRef* = rdflib.term.URIRef('https://schema.org/PodcastSeason')

PodcastSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/PodcastSeries')

Podiatric: *URIRef* = rdflib.term.URIRef('https://schema.org/Podiatric')

PoliceStation: *URIRef* = rdflib.term.URIRef('https://schema.org/PoliceStation')

Pond: *URIRef* = rdflib.term.URIRef('https://schema.org/Pond')

PostOffice: *URIRef* = rdflib.term.URIRef('https://schema.org/PostOffice')

PostalAddress: *URIRef* = rdflib.term.URIRef('https://schema.org/PostalAddress')

PostalCodeRangeSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/PostalCodeRangeSpecification')

Poster: *URIRef* = rdflib.term.URIRef('https://schema.org/Poster')

PotentialActionStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/PotentialActionStatus')

PreOrder: *URIRef* = rdflib.term.URIRef('https://schema.org/PreOrder')

PreOrderAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PreOrderAction')

PreSale: *URIRef* = rdflib.term.URIRef('https://schema.org/PreSale')

PregnancyHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/PregnancyHealthAspect')

PrependAction: *URIRef* = rdflib.term.URIRef('https://schema.org/PrependAction')

Preschool: *URIRef* = rdflib.term.URIRef('https://schema.org/Preschool')

PrescriptionOnly: *URIRef* = rdflib.term.URIRef('https://schema.org/PrescriptionOnly')

PresentationDigitalDocument: *URIRef* = rdflib.term.URIRef('https://schema.org/PresentationDigitalDocument')

PreventionHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/PreventionHealthAspect')

PreventionIndication: *URIRef* = rdflib.term.URIRef('https://schema.org/PreventionIndication')

PriceComponentTypeEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/PriceComponentTypeEnumeration')

PriceSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/PriceSpecification')

PriceTypeEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/PriceTypeEnumeration')

PrimaryCare: *URIRef* = rdflib.term.URIRef('https://schema.org/PrimaryCare')

Prion: *URIRef* = rdflib.term.URIRef('https://schema.org/Prion')

Product: *URIRef* = rdflib.term.URIRef('https://schema.org/Product')

ProductCollection: *URIRef* = rdflib.term.URIRef('https://schema.org/ProductCollection')

ProductGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/ProductGroup')

ProductModel: *URIRef* = rdflib.term.URIRef('https://schema.org/ProductModel')

ProfessionalService: *URIRef* = rdflib.term.URIRef('https://schema.org/ProfessionalService')

ProfilePage: *URIRef* = rdflib.term.URIRef('https://schema.org/ProfilePage')

PrognosisHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/PrognosisHealthAspect')

ProgramMembership: *URIRef* = rdflib.term.URIRef('https://schema.org/ProgramMembership')

Project: *URIRef* = rdflib.term.URIRef('https://schema.org/Project')

PronounceableText: *URIRef* = rdflib.term.URIRef('https://schema.org/PronounceableText')

Property: *URIRef* = rdflib.term.URIRef('https://schema.org/Property')

PropertyValue: *URIRef* = rdflib.term.URIRef('https://schema.org/PropertyValue')

PropertyValueSpecification: *URIRef* =
rdflib.term.URIRef('https://schema.org/PropertyValueSpecification')

Protein: *URIRef* = rdflib.term.URIRef('https://schema.org/Protein')

Protozoa: *URIRef* = rdflib.term.URIRef('https://schema.org/Protozoa')

Psychiatric: *URIRef* = rdflib.term.URIRef('https://schema.org/Psychiatric')

PsychologicalTreatment: *URIRef* =
rdflib.term.URIRef('https://schema.org/PsychologicalTreatment')

PublicHealth: *URIRef* = rdflib.term.URIRef('https://schema.org/PublicHealth')

PublicHolidays: *URIRef* = rdflib.term.URIRef('https://schema.org/PublicHolidays')

PublicSwimmingPool: *URIRef* =
rdflib.term.URIRef('https://schema.org/PublicSwimmingPool')

PublicToilet: *URIRef* = rdflib.term.URIRef('https://schema.org/PublicToilet')

PublicationEvent: *URIRef* =
rdflib.term.URIRef('https://schema.org/PublicationEvent')

PublicationIssue: *URIRef* =
rdflib.term.URIRef('https://schema.org/PublicationIssue')

PublicationVolume: *URIRef* =
rdflib.term.URIRef('https://schema.org/PublicationVolume')

Pulmonary: *URIRef* = rdflib.term.URIRef('https://schema.org/Pulmonary')

QAPage: *URIRef* = rdflib.term.URIRef('https://schema.org/QAPage')

QualitativeValue: *URIRef* =
rdflib.term.URIRef('https://schema.org/QualitativeValue')

QuantitativeValue: *URIRef* =
rdflib.term.URIRef('https://schema.org/QuantitativeValue')

QuantitativeValueDistribution: *URIRef* =
rdflib.term.URIRef('https://schema.org/QuantitativeValueDistribution')

Quantity: *URIRef* = rdflib.term.URIRef('https://schema.org/Quantity')

Question: *URIRef* = rdflib.term.URIRef('https://schema.org/Question')

Quiz: *URIRef* = rdflib.term.URIRef('https://schema.org/Quiz')

Quotation: *URIRef* = rdflib.term.URIRef('https://schema.org/Quotation')

QuoteAction: *URIRef* = rdflib.term.URIRef('https://schema.org/QuoteAction')

RVPark: *URIRef* = rdflib.term.URIRef('https://schema.org/RVPark')

RadiationTherapy: *URIRef* = rdflib.term.URIRef('https://schema.org/RadiationTherapy')

RadioBroadcastService: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioBroadcastService')

RadioChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioChannel')

RadioClip: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioClip')

RadioEpisode: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioEpisode')

RadioSeason: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioSeason')

RadioSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioSeries')

RadioStation: *URIRef* = rdflib.term.URIRef('https://schema.org/RadioStation')

Radiography: *URIRef* = rdflib.term.URIRef('https://schema.org/Radiography')

RandomizedTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/RandomizedTrial')

Rating: *URIRef* = rdflib.term.URIRef('https://schema.org/Rating')

ReactAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReactAction')

ReadAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReadAction')

ReadPermission: *URIRef* = rdflib.term.URIRef('https://schema.org/ReadPermission')

RealEstateAgent: *URIRef* = rdflib.term.URIRef('https://schema.org/RealEstateAgent')

RealEstateListing: *URIRef* = rdflib.term.URIRef('https://schema.org/RealEstateListing')

RearWheelDriveConfiguration: *URIRef* = rdflib.term.URIRef('https://schema.org/RearWheelDriveConfiguration')

ReceiveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReceiveAction')

Recipe: *URIRef* = rdflib.term.URIRef('https://schema.org/Recipe')

Recommendation: *URIRef* = rdflib.term.URIRef('https://schema.org/Recommendation')

RecommendedDoseSchedule: *URIRef* = rdflib.term.URIRef('https://schema.org/RecommendedDoseSchedule')

Recruiting: *URIRef* = rdflib.term.URIRef('https://schema.org/Recruiting')

RecyclingCenter: *URIRef* = rdflib.term.URIRef('https://schema.org/RecyclingCenter')

RefundTypeEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/RefundTypeEnumeration')

RefurbishedCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/RefurbishedCondition')

RegisterAction: *URIRef* = rdflib.term.URIRef('https://schema.org/RegisterAction')

Registry: *URIRef* = rdflib.term.URIRef('https://schema.org/Registry')

ReimbursementCap: *URIRef* =
rdflib.term.URIRef('https://schema.org/ReimbursementCap')

RejectAction: *URIRef* = rdflib.term.URIRef('https://schema.org/RejectAction')

RelatedTopicsHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/RelatedTopicsHealthAspect')

RemixAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/RemixAlbum')

Renal: *URIRef* = rdflib.term.URIRef('https://schema.org/Renal')

RentAction: *URIRef* = rdflib.term.URIRef('https://schema.org/RentAction')

RentalCarReservation: *URIRef* =
rdflib.term.URIRef('https://schema.org/RentalCarReservation')

RentalVehicleUsage: *URIRef* =
rdflib.term.URIRef('https://schema.org/RentalVehicleUsage')

RepaymentSpecification: *URIRef* =
rdflib.term.URIRef('https://schema.org/RepaymentSpecification')

ReplaceAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReplaceAction')

ReplyAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReplyAction')

Report: *URIRef* = rdflib.term.URIRef('https://schema.org/Report')

ReportageNewsArticle: *URIRef* =
rdflib.term.URIRef('https://schema.org/ReportageNewsArticle')

ReportedDoseSchedule: *URIRef* =
rdflib.term.URIRef('https://schema.org/ReportedDoseSchedule')

ResearchOrganization: *URIRef* =
rdflib.term.URIRef('https://schema.org/ResearchOrganization')

ResearchProject: *URIRef* = rdflib.term.URIRef('https://schema.org/ResearchProject')

Researcher: *URIRef* = rdflib.term.URIRef('https://schema.org/Researcher')

Reservation: *URIRef* = rdflib.term.URIRef('https://schema.org/Reservation')

ReservationCancelled: *URIRef* =
rdflib.term.URIRef('https://schema.org/ReservationCancelled')

ReservationConfirmed: *URIRef* =
rdflib.term.URIRef('https://schema.org/ReservationConfirmed')

ReservationHold: *URIRef* = rdflib.term.URIRef('https://schema.org/ReservationHold')

ReservationPackage: *URIRef* =
rdflib.term.URIRef('https://schema.org/ReservationPackage')

ReservationPending: *URIRef* = rdflib.term.URIRef('https://schema.org/ReservationPending')

ReservationStatusType: *URIRef* = rdflib.term.URIRef('https://schema.org/ReservationStatusType')

ReserveAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReserveAction')

Reservoir: *URIRef* = rdflib.term.URIRef('https://schema.org/Reservoir')

Residence: *URIRef* = rdflib.term.URIRef('https://schema.org/Residence')

Resort: *URIRef* = rdflib.term.URIRef('https://schema.org/Resort')

RespiratoryTherapy: *URIRef* = rdflib.term.URIRef('https://schema.org/RespiratoryTherapy')

Restaurant: *URIRef* = rdflib.term.URIRef('https://schema.org/Restaurant')

RestockingFees: *URIRef* = rdflib.term.URIRef('https://schema.org/RestockingFees')

RestrictedDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/RestrictedDiet')

ResultsAvailable: *URIRef* = rdflib.term.URIRef('https://schema.org/ResultsAvailable')

ResultsNotAvailable: *URIRef* = rdflib.term.URIRef('https://schema.org/ResultsNotAvailable')

ResumeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ResumeAction')

Retail: *URIRef* = rdflib.term.URIRef('https://schema.org/Retail')

ReturnAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnAction')

ReturnAtKiosk: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnAtKiosk')

ReturnByMail: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnByMail')

ReturnFeesCustomerResponsibility: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnFeesCustomerResponsibility')

ReturnFeesEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnFeesEnumeration')

ReturnInStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnInStore')

ReturnLabelCustomerResponsibility: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnLabelCustomerResponsibility')

ReturnLabelDownloadAndPrint: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnLabelDownloadAndPrint')

ReturnLabelInBox: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnLabelInBox')

ReturnLabelSourceEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnLabelSourceEnumeration')

ReturnMethodEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnMethodEnumeration')

ReturnShippingFees: *URIRef* = rdflib.term.URIRef('https://schema.org/ReturnShippingFees')

Review: *URIRef* = rdflib.term.URIRef('https://schema.org/Review')

ReviewAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ReviewAction')

ReviewNewsArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/ReviewNewsArticle')

Rheumatologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Rheumatologic')

RightHandDriving: *URIRef* = rdflib.term.URIRef('https://schema.org/RightHandDriving')

RisksOrComplicationsHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/RisksOrComplicationsHealthAspect')

RiverBodyOfWater: *URIRef* = rdflib.term.URIRef('https://schema.org/RiverBodyOfWater')

Role: *URIRef* = rdflib.term.URIRef('https://schema.org/Role')

RoofingContractor: *URIRef* = rdflib.term.URIRef('https://schema.org/RoofingContractor')

Room: *URIRef* = rdflib.term.URIRef('https://schema.org/Room')

RsvpAction: *URIRef* = rdflib.term.URIRef('https://schema.org/RsvpAction')

RsvpResponseMaybe: *URIRef* = rdflib.term.URIRef('https://schema.org/RsvpResponseMaybe')

RsvpResponseNo: *URIRef* = rdflib.term.URIRef('https://schema.org/RsvpResponseNo')

RsvpResponseType: *URIRef* = rdflib.term.URIRef('https://schema.org/RsvpResponseType')

RsvpResponseYes: *URIRef* = rdflib.term.URIRef('https://schema.org/RsvpResponseYes')

SRP: *URIRef* = rdflib.term.URIRef('https://schema.org/SRP')

SafetyHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/SafetyHealthAspect')

SaleEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/SaleEvent')

SalePrice: *URIRef* = rdflib.term.URIRef('https://schema.org/SalePrice')

SatireOrParodyContent: *URIRef* = rdflib.term.URIRef('https://schema.org/SatireOrParodyContent')

SatiricalArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/SatiricalArticle')

Saturday: *URIRef* = rdflib.term.URIRef('https://schema.org/Saturday')

Schedule: *URIRef* = rdflib.term.URIRef('https://schema.org/Schedule')

ScheduleAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ScheduleAction')

ScholarlyArticle: *URIRef* =
rdflib.term.URIRef('https://schema.org/ScholarlyArticle')

School: *URIRef* = rdflib.term.URIRef('https://schema.org/School')

SchoolDistrict: *URIRef* = rdflib.term.URIRef('https://schema.org/SchoolDistrict')

ScreeningEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/ScreeningEvent')

ScreeningHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/ScreeningHealthAspect')

Sculpture: *URIRef* = rdflib.term.URIRef('https://schema.org/Sculpture')

SeaBodyOfWater: *URIRef* = rdflib.term.URIRef('https://schema.org/SeaBodyOfWater')

SearchAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SearchAction')

SearchResultsPage: *URIRef* =
rdflib.term.URIRef('https://schema.org/SearchResultsPage')

Season: *URIRef* = rdflib.term.URIRef('https://schema.org/Season')

Seat: *URIRef* = rdflib.term.URIRef('https://schema.org/Seat')

SeatingMap: *URIRef* = rdflib.term.URIRef('https://schema.org/SeatingMap')

SeeDoctorHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/SeeDoctorHealthAspect')

SeekToAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SeekToAction')

SelfCareHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/SelfCareHealthAspect')

SelfStorage: *URIRef* = rdflib.term.URIRef('https://schema.org/SelfStorage')

SellAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SellAction')

SendAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SendAction')

Series: *URIRef* = rdflib.term.URIRef('https://schema.org/Series')

Service: *URIRef* = rdflib.term.URIRef('https://schema.org/Service')

ServiceChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/ServiceChannel')

ShareAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ShareAction')

SheetMusic: *URIRef* = rdflib.term.URIRef('https://schema.org/SheetMusic')

ShippingDeliveryTime: *URIRef* =
rdflib.term.URIRef('https://schema.org/ShippingDeliveryTime')

ShippingRateSettings: *URIRef* = rdflib.term.URIRef('https://schema.org/ShippingRateSettings')

ShoeStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ShoeStore')

ShoppingCenter: *URIRef* = rdflib.term.URIRef('https://schema.org/ShoppingCenter')

ShortStory: *URIRef* = rdflib.term.URIRef('https://schema.org/ShortStory')

SideEffectsHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/SideEffectsHealthAspect')

SingleBlindedTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/SingleBlindedTrial')

SingleCenterTrial: *URIRef* = rdflib.term.URIRef('https://schema.org/SingleCenterTrial')

SingleFamilyResidence: *URIRef* = rdflib.term.URIRef('https://schema.org/SingleFamilyResidence')

SinglePlayer: *URIRef* = rdflib.term.URIRef('https://schema.org/SinglePlayer')

SingleRelease: *URIRef* = rdflib.term.URIRef('https://schema.org/SingleRelease')

SiteNavigationElement: *URIRef* = rdflib.term.URIRef('https://schema.org/SiteNavigationElement')

SizeGroupEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/SizeGroupEnumeration')

SizeSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/SizeSpecification')

SizeSystemEnumeration: *URIRef* = rdflib.term.URIRef('https://schema.org/SizeSystemEnumeration')

SizeSystemImperial: *URIRef* = rdflib.term.URIRef('https://schema.org/SizeSystemImperial')

SizeSystemMetric: *URIRef* = rdflib.term.URIRef('https://schema.org/SizeSystemMetric')

SkiResort: *URIRef* = rdflib.term.URIRef('https://schema.org/SkiResort')

Skin: *URIRef* = rdflib.term.URIRef('https://schema.org/Skin')

SocialEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/SocialEvent')

SocialMediaPosting: *URIRef* = rdflib.term.URIRef('https://schema.org/SocialMediaPosting')

SoftwareApplication: *URIRef* = rdflib.term.URIRef('https://schema.org/SoftwareApplication')

SoftwareSourceCode: *URIRef* = rdflib.term.URIRef('https://schema.org/SoftwareSourceCode')

SoldOut: *URIRef* = rdflib.term.URIRef('https://schema.org/SoldOut')

SolveMathAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SolveMathAction')

SomeProducts: *URIRef* = rdflib.term.URIRef('https://schema.org/SomeProducts')

SoundtrackAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/SoundtrackAlbum')

SpeakableSpecification: *URIRef* =
rdflib.term.URIRef('https://schema.org/SpeakableSpecification')

SpecialAnnouncement: *URIRef* =
rdflib.term.URIRef('https://schema.org/SpecialAnnouncement')

Specialty: *URIRef* = rdflib.term.URIRef('https://schema.org/Specialty')

SpeechPathology: *URIRef* = rdflib.term.URIRef('https://schema.org/SpeechPathology')

SpokenWordAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/SpokenWordAlbum')

SportingGoodsStore: *URIRef* =
rdflib.term.URIRef('https://schema.org/SportingGoodsStore')

SportsActivityLocation: *URIRef* =
rdflib.term.URIRef('https://schema.org/SportsActivityLocation')

SportsClub: *URIRef* = rdflib.term.URIRef('https://schema.org/SportsClub')

SportsEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/SportsEvent')

SportsOrganization: *URIRef* =
rdflib.term.URIRef('https://schema.org/SportsOrganization')

SportsTeam: *URIRef* = rdflib.term.URIRef('https://schema.org/SportsTeam')

SpreadsheetDigitalDocument: *URIRef* =
rdflib.term.URIRef('https://schema.org/SpreadsheetDigitalDocument')

StadiumOrArena: *URIRef* = rdflib.term.URIRef('https://schema.org/StadiumOrArena')

StagedContent: *URIRef* = rdflib.term.URIRef('https://schema.org/StagedContent')

StagesHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/StagesHealthAspect')

State: *URIRef* = rdflib.term.URIRef('https://schema.org/State')

Statement: *URIRef* = rdflib.term.URIRef('https://schema.org/Statement')

StatisticalPopulation: *URIRef* =
rdflib.term.URIRef('https://schema.org/StatisticalPopulation')

StatusEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/StatusEnumeration')

SteeringPositionValue: *URIRef* =
rdflib.term.URIRef('https://schema.org/SteeringPositionValue')

Store: *URIRef* = rdflib.term.URIRef('https://schema.org/Store')

StoreCreditRefund: *URIRef* = rdflib.term.URIRef('https://schema.org/StoreCreditRefund')

StrengthTraining: *URIRef* = rdflib.term.URIRef('https://schema.org/StrengthTraining')

StructuredValue: *URIRef* = rdflib.term.URIRef('https://schema.org/StructuredValue')

StudioAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/StudioAlbum')

SubscribeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SubscribeAction')

Subscription: *URIRef* = rdflib.term.URIRef('https://schema.org/Subscription')

Substance: *URIRef* = rdflib.term.URIRef('https://schema.org/Substance')

SubwayStation: *URIRef* = rdflib.term.URIRef('https://schema.org/SubwayStation')

Suite: *URIRef* = rdflib.term.URIRef('https://schema.org/Suite')

Sunday: *URIRef* = rdflib.term.URIRef('https://schema.org/Sunday')

SuperficialAnatomy: *URIRef* = rdflib.term.URIRef('https://schema.org/SuperficialAnatomy')

Surgical: *URIRef* = rdflib.term.URIRef('https://schema.org/Surgical')

SurgicalProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/SurgicalProcedure')

SuspendAction: *URIRef* = rdflib.term.URIRef('https://schema.org/SuspendAction')

Suspended: *URIRef* = rdflib.term.URIRef('https://schema.org/Suspended')

SymptomsHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/SymptomsHealthAspect')

Synagogue: *URIRef* = rdflib.term.URIRef('https://schema.org/Synagogue')

TVClip: *URIRef* = rdflib.term.URIRef('https://schema.org/TVClip')

TVEpisode: *URIRef* = rdflib.term.URIRef('https://schema.org/TVEpisode')

TVSeason: *URIRef* = rdflib.term.URIRef('https://schema.org/TVSeason')

TVSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/TVSeries')

Table: *URIRef* = rdflib.term.URIRef('https://schema.org/Table')

TakeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TakeAction')

TattooParlor: *URIRef* = rdflib.term.URIRef('https://schema.org/TattooParlor')

Taxi: *URIRef* = rdflib.term.URIRef('https://schema.org/Taxi')

TaxiReservation: *URIRef* = rdflib.term.URIRef('https://schema.org/TaxiReservation')

TaxiService: *URIRef* = rdflib.term.URIRef('https://schema.org/TaxiService')

TaxiStand: *URIRef* = rdflib.term.URIRef('https://schema.org/TaxiStand')

TaxiVehicleUsage: *URIRef* = rdflib.term.URIRef('https://schema.org/TaxiVehicleUsage')

Taxon: *URIRef* = rdflib.term.URIRef('https://schema.org/Taxon')

TechArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/TechArticle')

TelevisionChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/TelevisionChannel')

TelevisionStation: *URIRef* = rdflib.term.URIRef('https://schema.org/TelevisionStation')

TennisComplex: *URIRef* = rdflib.term.URIRef('https://schema.org/TennisComplex')

Terminated: *URIRef* = rdflib.term.URIRef('https://schema.org/Terminated')

Text: *URIRef* = rdflib.term.URIRef('https://schema.org/Text')

TextDigitalDocument: *URIRef* = rdflib.term.URIRef('https://schema.org/TextDigitalDocument')

TheaterEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/TheaterEvent')

TheaterGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/TheaterGroup')

Therapeutic: *URIRef* = rdflib.term.URIRef('https://schema.org/Therapeutic')

TherapeuticProcedure: *URIRef* = rdflib.term.URIRef('https://schema.org/TherapeuticProcedure')

Thesis: *URIRef* = rdflib.term.URIRef('https://schema.org/Thesis')

Thing: *URIRef* = rdflib.term.URIRef('https://schema.org/Thing')

Throat: *URIRef* = rdflib.term.URIRef('https://schema.org/Throat')

Thursday: *URIRef* = rdflib.term.URIRef('https://schema.org/Thursday')

Ticket: *URIRef* = rdflib.term.URIRef('https://schema.org/Ticket')

TieAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TieAction')

Time: *URIRef* = rdflib.term.URIRef('https://schema.org/Time')

TipAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TipAction')

TireShop: *URIRef* = rdflib.term.URIRef('https://schema.org/TireShop')

TollFree: *URIRef* = rdflib.term.URIRef('https://schema.org/TollFree')

TouristAttraction: *URIRef* = rdflib.term.URIRef('https://schema.org/TouristAttraction')

TouristDestination: *URIRef* = rdflib.term.URIRef('https://schema.org/TouristDestination')

TouristInformationCenter: *URIRef* = rdflib.term.URIRef('https://schema.org/TouristInformationCenter')

TouristTrip: *URIRef* = rdflib.term.URIRef('https://schema.org/TouristTrip')

Toxicologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Toxicologic')

ToyStore: *URIRef* = rdflib.term.URIRef('https://schema.org/ToyStore')

TrackAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TrackAction')

TradeAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TradeAction')

TraditionalChinese: *URIRef* =
rdflib.term.URIRef('https://schema.org/TraditionalChinese')

TrainReservation: *URIRef* =
rdflib.term.URIRef('https://schema.org/TrainReservation')

TrainStation: *URIRef* = rdflib.term.URIRef('https://schema.org/TrainStation')

TrainTrip: *URIRef* = rdflib.term.URIRef('https://schema.org/TrainTrip')

TransferAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TransferAction')

TransformedContent: *URIRef* =
rdflib.term.URIRef('https://schema.org/TransformedContent')

TransitMap: *URIRef* = rdflib.term.URIRef('https://schema.org/TransitMap')

TravelAction: *URIRef* = rdflib.term.URIRef('https://schema.org/TravelAction')

TravelAgency: *URIRef* = rdflib.term.URIRef('https://schema.org/TravelAgency')

TreatmentIndication: *URIRef* =
rdflib.term.URIRef('https://schema.org/TreatmentIndication')

TreatmentsHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/TreatmentsHealthAspect')

Trip: *URIRef* = rdflib.term.URIRef('https://schema.org/Trip')

TripleBlindedTrial: *URIRef* =
rdflib.term.URIRef('https://schema.org/TripleBlindedTrial')

Tuesday: *URIRef* = rdflib.term.URIRef('https://schema.org/Tuesday')

TypeAndQuantityNode: *URIRef* =
rdflib.term.URIRef('https://schema.org/TypeAndQuantityNode')

TypesHealthAspect: *URIRef* =
rdflib.term.URIRef('https://schema.org/TypesHealthAspect')

UKNonprofitType: *URIRef* = rdflib.term.URIRef('https://schema.org/UKNonprofitType')

UKTrust: *URIRef* = rdflib.term.URIRef('https://schema.org/UKTrust')

URL: *URIRef* = rdflib.term.URIRef('https://schema.org/URL')

USNonprofitType: *URIRef* = rdflib.term.URIRef('https://schema.org/USNonprofitType')

Ultrasound: *URIRef* = rdflib.term.URIRef('https://schema.org/Ultrasound')

UnRegisterAction: *URIRef* = rdflib.term.URIRef('https://schema.org/UnRegisterAction')

UnemploymentSupport: *URIRef* = rdflib.term.URIRef('https://schema.org/UnemploymentSupport')

UnincorporatedAssociationCharity: *URIRef* = rdflib.term.URIRef('https://schema.org/UnincorporatedAssociationCharity')

UnitPriceSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/UnitPriceSpecification')

UnofficialLegalValue: *URIRef* = rdflib.term.URIRef('https://schema.org/UnofficialLegalValue')

UpdateAction: *URIRef* = rdflib.term.URIRef('https://schema.org/UpdateAction')

Urologic: *URIRef* = rdflib.term.URIRef('https://schema.org/Urologic')

UsageOrScheduleHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/UsageOrScheduleHealthAspect')

UseAction: *URIRef* = rdflib.term.URIRef('https://schema.org/UseAction')

UsedCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/UsedCondition')

UserBlocks: *URIRef* = rdflib.term.URIRef('https://schema.org/UserBlocks')

UserCheckins: *URIRef* = rdflib.term.URIRef('https://schema.org/UserCheckins')

UserComments: *URIRef* = rdflib.term.URIRef('https://schema.org/UserComments')

UserDownloads: *URIRef* = rdflib.term.URIRef('https://schema.org/UserDownloads')

UserInteraction: *URIRef* = rdflib.term.URIRef('https://schema.org/UserInteraction')

UserLikes: *URIRef* = rdflib.term.URIRef('https://schema.org/UserLikes')

UserPageVisits: *URIRef* = rdflib.term.URIRef('https://schema.org/UserPageVisits')

UserPlays: *URIRef* = rdflib.term.URIRef('https://schema.org/UserPlays')

UserPlusOnes: *URIRef* = rdflib.term.URIRef('https://schema.org/UserPlusOnes')

UserReview: *URIRef* = rdflib.term.URIRef('https://schema.org/UserReview')

UserTweets: *URIRef* = rdflib.term.URIRef('https://schema.org/UserTweets')

VeganDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/VeganDiet')

VegetarianDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/VegetarianDiet')

Vehicle: *URIRef* = rdflib.term.URIRef('https://schema.org/Vehicle')

Vein: *URIRef* = rdflib.term.URIRef('https://schema.org/Vein')

VenueMap: *URIRef* = rdflib.term.URIRef('https://schema.org/VenueMap')

Vessel: *URIRef* = rdflib.term.URIRef('https://schema.org/Vessel')

VeterinaryCare: *URIRef* = rdflib.term.URIRef('https://schema.org/VeterinaryCare')

VideoGallery: *URIRef* = rdflib.term.URIRef('https://schema.org/VideoGallery')

VideoGame: *URIRef* = rdflib.term.URIRef('https://schema.org/VideoGame')

VideoGameClip: *URIRef* = rdflib.term.URIRef('https://schema.org/VideoGameClip')

VideoGameSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/VideoGameSeries')

VideoObject: *URIRef* = rdflib.term.URIRef('https://schema.org/VideoObject')

VideoObjectSnapshot: *URIRef* = rdflib.term.URIRef('https://schema.org/VideoObjectSnapshot')

ViewAction: *URIRef* = rdflib.term.URIRef('https://schema.org/ViewAction')

VinylFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/VinylFormat')

VirtualLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/VirtualLocation')

Virus: *URIRef* = rdflib.term.URIRef('https://schema.org/Virus')

VisualArtsEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/VisualArtsEvent')

VisualArtwork: *URIRef* = rdflib.term.URIRef('https://schema.org/VisualArtwork')

VitalSign: *URIRef* = rdflib.term.URIRef('https://schema.org/VitalSign')

Volcano: *URIRef* = rdflib.term.URIRef('https://schema.org/Volcano')

VoteAction: *URIRef* = rdflib.term.URIRef('https://schema.org/VoteAction')

WPAdBlock: *URIRef* = rdflib.term.URIRef('https://schema.org/WPAdBlock')

WPFooter: *URIRef* = rdflib.term.URIRef('https://schema.org/WPFooter')

WPHeader: *URIRef* = rdflib.term.URIRef('https://schema.org/WPHeader')

WPSideBar: *URIRef* = rdflib.term.URIRef('https://schema.org/WPSideBar')

WantAction: *URIRef* = rdflib.term.URIRef('https://schema.org/WantAction')

WarrantyPromise: *URIRef* = rdflib.term.URIRef('https://schema.org/WarrantyPromise')

WarrantyScope: *URIRef* = rdflib.term.URIRef('https://schema.org/WarrantyScope')

WatchAction: *URIRef* = rdflib.term.URIRef('https://schema.org/WatchAction')

Waterfall: *URIRef* = rdflib.term.URIRef('https://schema.org/Waterfall')

WearAction: *URIRef* = rdflib.term.URIRef('https://schema.org/WearAction')

WearableMeasurementBack: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableMeasurementBack')

WearableMeasurementChestOrBust: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableMeasurementChestOrBust')

WearableMeasurementCollar: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementCollar')

WearableMeasurementCup: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementCup')

WearableMeasurementHeight: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementHeight')

WearableMeasurementHips: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementHips')

WearableMeasurementInseam: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementInseam')

WearableMeasurementLength: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementLength')

WearableMeasurementOutsideLeg: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementOutsideLeg')

WearableMeasurementSleeve: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementSleeve')

WearableMeasurementTypeEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementTypeEnumeration')

WearableMeasurementWaist: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementWaist')

WearableMeasurementWidth: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableMeasurementWidth')

WearableSizeGroupBig: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupBig')

WearableSizeGroupBoys: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupBoys')

WearableSizeGroupEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupEnumeration')

WearableSizeGroupExtraShort: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupExtraShort')

WearableSizeGroupExtraTall: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupExtraTall')

WearableSizeGroupGirls: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupGirls')

WearableSizeGroupHusky: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupHusky')

WearableSizeGroupInfants: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupInfants')

WearableSizeGroupJuniors: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupJuniors')

WearableSizeGroupMaternity: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupMaternity')

WearableSizeGroupMens: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupMens')

WearableSizeGroupMisses: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupMisses')

WearableSizeGroupPetite: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupPetite')

WearableSizeGroupPlus: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupPlus')

WearableSizeGroupRegular: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupRegular')

WearableSizeGroupShort: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupShort')

WearableSizeGroupTall: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupTall')

WearableSizeGroupWomens: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeGroupWomens')

WearableSizeSystemAU: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemAU')

WearableSizeSystemBR: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemBR')

WearableSizeSystemCN: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemCN')

WearableSizeSystemContinental: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemContinental')

WearableSizeSystemDE: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemDE')

WearableSizeSystemEN13402: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemEN13402')

WearableSizeSystemEnumeration: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemEnumeration')

WearableSizeSystemEurope: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemEurope')

WearableSizeSystemFR: *URIRef* =
rdflib.term.URIRef('https://schema.org/WearableSizeSystemFR')

WearableSizeSystemGS1: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableSizeSystemGS1')

WearableSizeSystemIT: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableSizeSystemIT')

WearableSizeSystemJP: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableSizeSystemJP')

WearableSizeSystemMX: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableSizeSystemMX')

WearableSizeSystemUK: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableSizeSystemUK')

WearableSizeSystemUS: *URIRef* = rdflib.term.URIRef('https://schema.org/WearableSizeSystemUS')

WebAPI: *URIRef* = rdflib.term.URIRef('https://schema.org/WebAPI')

WebApplication: *URIRef* = rdflib.term.URIRef('https://schema.org/WebApplication')

WebContent: *URIRef* = rdflib.term.URIRef('https://schema.org/WebContent')

WebPage: *URIRef* = rdflib.term.URIRef('https://schema.org/WebPage')

WebPageElement: *URIRef* = rdflib.term.URIRef('https://schema.org/WebPageElement')

WebSite: *URIRef* = rdflib.term.URIRef('https://schema.org/WebSite')

Wednesday: *URIRef* = rdflib.term.URIRef('https://schema.org/Wednesday')

WesternConventional: *URIRef* = rdflib.term.URIRef('https://schema.org/WesternConventional')

Wholesale: *URIRef* = rdflib.term.URIRef('https://schema.org/Wholesale')

WholesaleStore: *URIRef* = rdflib.term.URIRef('https://schema.org/WholesaleStore')

WinAction: *URIRef* = rdflib.term.URIRef('https://schema.org/WinAction')

Winery: *URIRef* = rdflib.term.URIRef('https://schema.org/Winery')

Withdrawn: *URIRef* = rdflib.term.URIRef('https://schema.org/Withdrawn')

WorkBasedProgram: *URIRef* = rdflib.term.URIRef('https://schema.org/WorkBasedProgram')

WorkersUnion: *URIRef* = rdflib.term.URIRef('https://schema.org/WorkersUnion')

WriteAction: *URIRef* = rdflib.term.URIRef('https://schema.org/WriteAction')

WritePermission: *URIRef* = rdflib.term.URIRef('https://schema.org/WritePermission')

XPathType: *URIRef* = rdflib.term.URIRef('https://schema.org/XPathType')

XRay: *URIRef* = rdflib.term.URIRef('https://schema.org/XRay')

ZoneBoardingPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/ZoneBoardingPolicy')

Zoo: *URIRef* = rdflib.term.URIRef('https://schema.org/Zoo')

about: *URIRef* = rdflib.term.URIRef('https://schema.org/about')

abridged: *URIRef* = rdflib.term.URIRef('https://schema.org/abridged')

abstract: *URIRef* = rdflib.term.URIRef('https://schema.org/abstract')

accelerationTime: *URIRef* = rdflib.term.URIRef('https://schema.org/accelerationTime')

acceptedAnswer: *URIRef* = rdflib.term.URIRef('https://schema.org/acceptedAnswer')

acceptedOffer: *URIRef* = rdflib.term.URIRef('https://schema.org/acceptedOffer')

acceptedPaymentMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/acceptedPaymentMethod')

acceptsReservations: *URIRef* = rdflib.term.URIRef('https://schema.org/acceptsReservations')

accessCode: *URIRef* = rdflib.term.URIRef('https://schema.org/accessCode')

accessMode: *URIRef* = rdflib.term.URIRef('https://schema.org/accessMode')

accessModeSufficient: *URIRef* = rdflib.term.URIRef('https://schema.org/accessModeSufficient')

accessibilityAPI: *URIRef* = rdflib.term.URIRef('https://schema.org/accessibilityAPI')

accessibilityControl: *URIRef* = rdflib.term.URIRef('https://schema.org/accessibilityControl')

accessibilityFeature: *URIRef* = rdflib.term.URIRef('https://schema.org/accessibilityFeature')

accessibilityHazard: *URIRef* = rdflib.term.URIRef('https://schema.org/accessibilityHazard')

accessibilitySummary: *URIRef* = rdflib.term.URIRef('https://schema.org/accessibilitySummary')

accommodationCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/accommodationCategory')

accommodationFloorPlan: *URIRef* = rdflib.term.URIRef('https://schema.org/accommodationFloorPlan')

accountId: *URIRef* = rdflib.term.URIRef('https://schema.org/accountId')

accountMinimumInflow: *URIRef* = rdflib.term.URIRef('https://schema.org/accountMinimumInflow')

accountOverdraftLimit: *URIRef* = rdflib.term.URIRef('https://schema.org/accountOverdraftLimit')

accountablePerson: *URIRef* = rdflib.term.URIRef('https://schema.org/accountablePerson')

acquireLicensePage: *URIRef* = rdflib.term.URIRef('https://schema.org/acquireLicensePage')

acquiredFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/acquiredFrom')

acrissCode: *URIRef* = rdflib.term.URIRef('https://schema.org/acrissCode')

actionAccessibilityRequirement: *URIRef* = rdflib.term.URIRef('https://schema.org/actionAccessibilityRequirement')

actionApplication: *URIRef* = rdflib.term.URIRef('https://schema.org/actionApplication')

actionOption: *URIRef* = rdflib.term.URIRef('https://schema.org/actionOption')

actionPlatform: *URIRef* = rdflib.term.URIRef('https://schema.org/actionPlatform')

actionStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/actionStatus')

actionableFeedbackPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/actionableFeedbackPolicy')

activeIngredient: *URIRef* = rdflib.term.URIRef('https://schema.org/activeIngredient')

activityDuration: *URIRef* = rdflib.term.URIRef('https://schema.org/activityDuration')

activityFrequency: *URIRef* = rdflib.term.URIRef('https://schema.org/activityFrequency')

actor: *URIRef* = rdflib.term.URIRef('https://schema.org/actor')

actors: *URIRef* = rdflib.term.URIRef('https://schema.org/actors')

addOn: *URIRef* = rdflib.term.URIRef('https://schema.org/addOn')

additionalName: *URIRef* = rdflib.term.URIRef('https://schema.org/additionalName')

additionalNumberOfGuests: *URIRef* = rdflib.term.URIRef('https://schema.org/additionalNumberOfGuests')

additionalProperty: *URIRef* = rdflib.term.URIRef('https://schema.org/additionalProperty')

additionalType: *URIRef* = rdflib.term.URIRef('https://schema.org/additionalType')

additionalVariable: *URIRef* = rdflib.term.URIRef('https://schema.org/additionalVariable')

address: *URIRef* = rdflib.term.URIRef('https://schema.org/address')

addressCountry: *URIRef* = rdflib.term.URIRef('https://schema.org/addressCountry')

addressLocality: *URIRef* = rdflib.term.URIRef('https://schema.org/addressLocality')

addressRegion: *URIRef* = rdflib.term.URIRef('https://schema.org/addressRegion')

administrationRoute: *URIRef* = rdflib.term.URIRef('https://schema.org/administrationRoute')

advanceBookingRequirement: *URIRef* = rdflib.term.URIRef('https://schema.org/advanceBookingRequirement')

adverseOutcome: *URIRef* = rdflib.term.URIRef('https://schema.org/adverseOutcome')

affectedBy: *URIRef* = rdflib.term.URIRef('https://schema.org/affectedBy')

affiliation: *URIRef* = rdflib.term.URIRef('https://schema.org/affiliation')

afterMedia: *URIRef* = rdflib.term.URIRef('https://schema.org/afterMedia')

agent: *URIRef* = rdflib.term.URIRef('https://schema.org/agent')

aggregateRating: *URIRef* = rdflib.term.URIRef('https://schema.org/aggregateRating')

aircraft: *URIRef* = rdflib.term.URIRef('https://schema.org/aircraft')

album: *URIRef* = rdflib.term.URIRef('https://schema.org/album')

albumProductionType: *URIRef* = rdflib.term.URIRef('https://schema.org/albumProductionType')

albumRelease: *URIRef* = rdflib.term.URIRef('https://schema.org/albumRelease')

albumReleaseType: *URIRef* = rdflib.term.URIRef('https://schema.org/albumReleaseType')

albums: *URIRef* = rdflib.term.URIRef('https://schema.org/albums')

alcoholWarning: *URIRef* = rdflib.term.URIRef('https://schema.org/alcoholWarning')

algorithm: *URIRef* = rdflib.term.URIRef('https://schema.org/algorithm')

alignmentType: *URIRef* = rdflib.term.URIRef('https://schema.org/alignmentType')

alternateName: *URIRef* = rdflib.term.URIRef('https://schema.org/alternateName')

alternativeHeadline: *URIRef* = rdflib.term.URIRef('https://schema.org/alternativeHeadline')

alternativeOf: *URIRef* = rdflib.term.URIRef('https://schema.org/alternativeOf')

alumni: *URIRef* = rdflib.term.URIRef('https://schema.org/alumni')

alumniOf: *URIRef* = rdflib.term.URIRef('https://schema.org/alumniOf')

amenityFeature: *URIRef* = rdflib.term.URIRef('https://schema.org/amenityFeature')

amount: *URIRef* = rdflib.term.URIRef('https://schema.org/amount')

amountOfThisGood: *URIRef* = rdflib.term.URIRef('https://schema.org/amountOfThisGood')

announcementLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/announcementLocation')

annualPercentageRate: *URIRef* = rdflib.term.URIRef('https://schema.org/annualPercentageRate')

answerCount: *URIRef* = rdflib.term.URIRef('https://schema.org/answerCount')

answerExplanation: *URIRef* = rdflib.term.URIRef('https://schema.org/answerExplanation')

antagonist: *URIRef* = rdflib.term.URIRef('https://schema.org/antagonist')

appearance: *URIRef* = rdflib.term.URIRef('https://schema.org/appearance')

applicableLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/applicableLocation')

applicantLocationRequirements: *URIRef* = rdflib.term.URIRef('https://schema.org/applicantLocationRequirements')

application: *URIRef* = rdflib.term.URIRef('https://schema.org/application')

applicationCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/applicationCategory')

applicationContact: *URIRef* = rdflib.term.URIRef('https://schema.org/applicationContact')

applicationDeadline: *URIRef* = rdflib.term.URIRef('https://schema.org/applicationDeadline')

applicationStartDate: *URIRef* = rdflib.term.URIRef('https://schema.org/applicationStartDate')

applicationSubCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/applicationSubCategory')

applicationSuite: *URIRef* = rdflib.term.URIRef('https://schema.org/applicationSuite')

appliesToDeliveryMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/appliesToDeliveryMethod')

appliesToPaymentMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/appliesToPaymentMethod')

archiveHeld: *URIRef* = rdflib.term.URIRef('https://schema.org/archiveHeld')

archivedAt: *URIRef* = rdflib.term.URIRef('https://schema.org/archivedAt')

area: *URIRef* = rdflib.term.URIRef('https://schema.org/area')

areaServed: *URIRef* = rdflib.term.URIRef('https://schema.org/areaServed')

arrivalAirport: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalAirport')

arrivalBoatTerminal: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalBoatTerminal')

arrivalBusStop: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalBusStop')

arrivalGate: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalGate')

arrivalPlatform: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalPlatform')

arrivalStation: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalStation')

arrivalTerminal: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalTerminal')

arrivalTime: *URIRef* = rdflib.term.URIRef('https://schema.org/arrivalTime')

artEdition: *URIRef* = rdflib.term.URIRef('https://schema.org/artEdition')

artMedium: *URIRef* = rdflib.term.URIRef('https://schema.org/artMedium')

arterialBranch: *URIRef* = rdflib.term.URIRef('https://schema.org/arterialBranch')

artform: *URIRef* = rdflib.term.URIRef('https://schema.org/artform')

articleBody: *URIRef* = rdflib.term.URIRef('https://schema.org/articleBody')

articleSection: *URIRef* = rdflib.term.URIRef('https://schema.org/articleSection')

artist: *URIRef* = rdflib.term.URIRef('https://schema.org/artist')

artworkSurface: *URIRef* = rdflib.term.URIRef('https://schema.org/artworkSurface')

aspect: *URIRef* = rdflib.term.URIRef('https://schema.org/aspect')

assembly: *URIRef* = rdflib.term.URIRef('https://schema.org/assembly')

assemblyVersion: *URIRef* = rdflib.term.URIRef('https://schema.org/assemblyVersion')

assesses: *URIRef* = rdflib.term.URIRef('https://schema.org/assesses')

associatedAnatomy: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedAnatomy')

associatedArticle: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedArticle')

associatedClaimReview: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedClaimReview')

associatedDisease: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedDisease')

associatedMedia: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedMedia')

associatedMediaReview: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedMediaReview')

associatedPathophysiology: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedPathophysiology')

associatedReview: *URIRef* = rdflib.term.URIRef('https://schema.org/associatedReview')

athlete: *URIRef* = rdflib.term.URIRef('https://schema.org/athlete')

attendee: *URIRef* = rdflib.term.URIRef('https://schema.org/attendee')

attendees: *URIRef* = rdflib.term.URIRef('https://schema.org/attendees')

audience: *URIRef* = rdflib.term.URIRef('https://schema.org/audience')

audienceType: *URIRef* = rdflib.term.URIRef('https://schema.org/audienceType')

audio: *URIRef* = rdflib.term.URIRef('https://schema.org/audio')

authenticator: *URIRef* = rdflib.term.URIRef('https://schema.org/authenticator')

author: *URIRef* = rdflib.term.URIRef('https://schema.org/author')

availability: *URIRef* = rdflib.term.URIRef('https://schema.org/availability')

availabilityEnds: *URIRef* = rdflib.term.URIRef('https://schema.org/availabilityEnds')

availabilityStarts: *URIRef* = rdflib.term.URIRef('https://schema.org/availabilityStarts')

availableAtOrFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/availableAtOrFrom')

availableChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/availableChannel')

availableDeliveryMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/availableDeliveryMethod')

availableFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/availableFrom')

availableIn: *URIRef* = rdflib.term.URIRef('https://schema.org/availableIn')

availableLanguage: *URIRef* = rdflib.term.URIRef('https://schema.org/availableLanguage')

availableOnDevice: *URIRef* = rdflib.term.URIRef('https://schema.org/availableOnDevice')

availableService: *URIRef* = rdflib.term.URIRef('https://schema.org/availableService')

availableStrength: *URIRef* = rdflib.term.URIRef('https://schema.org/availableStrength')

availableTest: *URIRef* = rdflib.term.URIRef('https://schema.org/availableTest')

availableThrough: *URIRef* = rdflib.term.URIRef('https://schema.org/availableThrough')

award: *URIRef* = rdflib.term.URIRef('https://schema.org/award')

awards: *URIRef* = rdflib.term.URIRef('https://schema.org/awards')

awayTeam: *URIRef* = rdflib.term.URIRef('https://schema.org/awayTeam')

backstory: *URIRef* = rdflib.term.URIRef('https://schema.org/backstory')

bankAccountType: *URIRef* = rdflib.term.URIRef('https://schema.org/bankAccountType')

baseSalary: *URIRef* = rdflib.term.URIRef('https://schema.org/baseSalary')

bccRecipient: *URIRef* = rdflib.term.URIRef('https://schema.org/bccRecipient')

bed: *URIRef* = rdflib.term.URIRef('https://schema.org/bed')

beforeMedia: *URIRef* = rdflib.term.URIRef('https://schema.org/beforeMedia')

beneficiaryBank: *URIRef* = rdflib.term.URIRef('https://schema.org/beneficiaryBank')

benefits: *URIRef* = rdflib.term.URIRef('https://schema.org/benefits')

benefitsSummaryUrl: *URIRef* =
rdflib.term.URIRef('https://schema.org/benefitsSummaryUrl')

bestRating: *URIRef* = rdflib.term.URIRef('https://schema.org/bestRating')

billingAddress: *URIRef* = rdflib.term.URIRef('https://schema.org/billingAddress')

billingDuration: *URIRef* = rdflib.term.URIRef('https://schema.org/billingDuration')

billingIncrement: *URIRef* =
rdflib.term.URIRef('https://schema.org/billingIncrement')

billingPeriod: *URIRef* = rdflib.term.URIRef('https://schema.org/billingPeriod')

billingStart: *URIRef* = rdflib.term.URIRef('https://schema.org/billingStart')

bioChemInteraction: *URIRef* =
rdflib.term.URIRef('https://schema.org/bioChemInteraction')

bioChemSimilarity: *URIRef* =
rdflib.term.URIRef('https://schema.org/bioChemSimilarity')

biologicalRole: *URIRef* = rdflib.term.URIRef('https://schema.org/biologicalRole')

biomechnicalClass: *URIRef* =
rdflib.term.URIRef('https://schema.org/biomechnicalClass')

birthDate: *URIRef* = rdflib.term.URIRef('https://schema.org/birthDate')

birthPlace: *URIRef* = rdflib.term.URIRef('https://schema.org/birthPlace')

bitrate: *URIRef* = rdflib.term.URIRef('https://schema.org/bitrate')

blogPost: *URIRef* = rdflib.term.URIRef('https://schema.org/blogPost')

blogPosts: *URIRef* = rdflib.term.URIRef('https://schema.org/blogPosts')

bloodSupply: *URIRef* = rdflib.term.URIRef('https://schema.org/bloodSupply')

boardingGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/boardingGroup')

boardingPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/boardingPolicy')

bodyLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/bodyLocation')

bodyType: *URIRef* = rdflib.term.URIRef('https://schema.org/bodyType')

bookEdition: *URIRef* = rdflib.term.URIRef('https://schema.org/bookEdition')

bookFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/bookFormat')

bookingAgent: *URIRef* = rdflib.term.URIRef('https://schema.org/bookingAgent')

bookingTime: *URIRef* = rdflib.term.URIRef('https://schema.org/bookingTime')

borrower: *URIRef* = rdflib.term.URIRef('https://schema.org/borrower')

box: *URIRef* = rdflib.term.URIRef('https://schema.org/box')

branch: *URIRef* = rdflib.term.URIRef('https://schema.org/branch')

branchCode: *URIRef* = rdflib.term.URIRef('https://schema.org/branchCode')

branchOf: *URIRef* = rdflib.term.URIRef('https://schema.org/branchOf')

brand: *URIRef* = rdflib.term.URIRef('https://schema.org/brand')

breadcrumb: *URIRef* = rdflib.term.URIRef('https://schema.org/breadcrumb')

breastfeedingWarning: *URIRef* =
rdflib.term.URIRef('https://schema.org/breastfeedingWarning')

broadcastAffiliateOf: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastAffiliateOf')

broadcastChannelId: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastChannelId')

broadcastDisplayName: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastDisplayName')

broadcastFrequency: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastFrequency')

broadcastFrequencyValue: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastFrequencyValue')

broadcastOfEvent: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastOfEvent')

broadcastServiceTier: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastServiceTier')

broadcastSignalModulation: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastSignalModulation')

broadcastSubChannel: *URIRef* =
rdflib.term.URIRef('https://schema.org/broadcastSubChannel')

broadcastTimezone: *URIRef* = rdflib.term.URIRef('https://schema.org/broadcastTimezone')

broadcaster: *URIRef* = rdflib.term.URIRef('https://schema.org/broadcaster')

broker: *URIRef* = rdflib.term.URIRef('https://schema.org/broker')

browserRequirements: *URIRef* = rdflib.term.URIRef('https://schema.org/browserRequirements')

busName: *URIRef* = rdflib.term.URIRef('https://schema.org/busName')

busNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/busNumber')

businessDays: *URIRef* = rdflib.term.URIRef('https://schema.org/businessDays')

businessFunction: *URIRef* = rdflib.term.URIRef('https://schema.org/businessFunction')

buyer: *URIRef* = rdflib.term.URIRef('https://schema.org/buyer')

byArtist: *URIRef* = rdflib.term.URIRef('https://schema.org/byArtist')

byDay: *URIRef* = rdflib.term.URIRef('https://schema.org/byDay')

byMonth: *URIRef* = rdflib.term.URIRef('https://schema.org/byMonth')

byMonthDay: *URIRef* = rdflib.term.URIRef('https://schema.org/byMonthDay')

byMonthWeek: *URIRef* = rdflib.term.URIRef('https://schema.org/byMonthWeek')

callSign: *URIRef* = rdflib.term.URIRef('https://schema.org/callSign')

calories: *URIRef* = rdflib.term.URIRef('https://schema.org/calories')

candidate: *URIRef* = rdflib.term.URIRef('https://schema.org/candidate')

caption: *URIRef* = rdflib.term.URIRef('https://schema.org/caption')

carbohydrateContent: *URIRef* = rdflib.term.URIRef('https://schema.org/carbohydrateContent')

cargoVolume: *URIRef* = rdflib.term.URIRef('https://schema.org/cargoVolume')

carrier: *URIRef* = rdflib.term.URIRef('https://schema.org/carrier')

carrierRequirements: *URIRef* = rdflib.term.URIRef('https://schema.org/carrierRequirements')

cashBack: *URIRef* = rdflib.term.URIRef('https://schema.org/cashBack')

catalog: *URIRef* = rdflib.term.URIRef('https://schema.org/catalog')

catalogNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/catalogNumber')

category: *URIRef* = rdflib.term.URIRef('https://schema.org/category')

causeOf: *URIRef* = rdflib.term.URIRef('https://schema.org/causeOf')

ccRecipient: *URIRef* = rdflib.term.URIRef('https://schema.org/ccRecipient')

character: *URIRef* = rdflib.term.URIRef('https://schema.org/character')

characterAttribute: *URIRef* = rdflib.term.URIRef('https://schema.org/characterAttribute')

characterName: *URIRef* = rdflib.term.URIRef('https://schema.org/characterName')

cheatCode: *URIRef* = rdflib.term.URIRef('https://schema.org/cheatCode')

checkinTime: *URIRef* = rdflib.term.URIRef('https://schema.org/checkinTime')

checkoutTime: *URIRef* = rdflib.term.URIRef('https://schema.org/checkoutTime')

chemicalComposition: *URIRef* = rdflib.term.URIRef('https://schema.org/chemicalComposition')

chemicalRole: *URIRef* = rdflib.term.URIRef('https://schema.org/chemicalRole')

childMaxAge: *URIRef* = rdflib.term.URIRef('https://schema.org/childMaxAge')

childMinAge: *URIRef* = rdflib.term.URIRef('https://schema.org/childMinAge')

childTaxon: *URIRef* = rdflib.term.URIRef('https://schema.org/childTaxon')

children: *URIRef* = rdflib.term.URIRef('https://schema.org/children')

cholesterolContent: *URIRef* = rdflib.term.URIRef('https://schema.org/cholesterolContent')

circle: *URIRef* = rdflib.term.URIRef('https://schema.org/circle')

citation: *URIRef* = rdflib.term.URIRef('https://schema.org/citation')

claimInterpreter: *URIRef* = rdflib.term.URIRef('https://schema.org/claimInterpreter')

claimReviewed: *URIRef* = rdflib.term.URIRef('https://schema.org/claimReviewed')

clincalPharmacology: *URIRef* = rdflib.term.URIRef('https://schema.org/clincalPharmacology')

clinicalPharmacology: *URIRef* = rdflib.term.URIRef('https://schema.org/clinicalPharmacology')

clipNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/clipNumber')

closes: *URIRef* = rdflib.term.URIRef('https://schema.org/closes')

coach: *URIRef* = rdflib.term.URIRef('https://schema.org/coach')

code: *URIRef* = rdflib.term.URIRef('https://schema.org/code')

codeRepository: *URIRef* = rdflib.term.URIRef('https://schema.org/codeRepository')

codeSampleType: *URIRef* = rdflib.term.URIRef('https://schema.org/codeSampleType')

codeValue: *URIRef* = rdflib.term.URIRef('https://schema.org/codeValue')

codingSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/codingSystem')

colleague: *URIRef* = rdflib.term.URIRef('https://schema.org/colleague')

colleagues: *URIRef* = rdflib.term.URIRef('https://schema.org/colleagues')

collection: *URIRef* = rdflib.term.URIRef('https://schema.org/collection')

collectionSize: *URIRef* = rdflib.term.URIRef('https://schema.org/collectionSize')

color: *URIRef* = rdflib.term.URIRef('https://schema.org/color')

colorist: *URIRef* = rdflib.term.URIRef('https://schema.org/colorist')

comment: *URIRef* = rdflib.term.URIRef('https://schema.org/comment')

commentCount: *URIRef* = rdflib.term.URIRef('https://schema.org/commentCount')

commentText: *URIRef* = rdflib.term.URIRef('https://schema.org/commentText')

commentTime: *URIRef* = rdflib.term.URIRef('https://schema.org/commentTime')

competencyRequired: *URIRef* = rdflib.term.URIRef('https://schema.org/competencyRequired')

competitor: *URIRef* = rdflib.term.URIRef('https://schema.org/competitor')

composer: *URIRef* = rdflib.term.URIRef('https://schema.org/composer')

comprisedOf: *URIRef* = rdflib.term.URIRef('https://schema.org/comprisedOf')

conditionsOfAccess: *URIRef* = rdflib.term.URIRef('https://schema.org/conditionsOfAccess')

confirmationNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/confirmationNumber')

connectedTo: *URIRef* = rdflib.term.URIRef('https://schema.org/connectedTo')

constrainingProperty: *URIRef* = rdflib.term.URIRef('https://schema.org/constrainingProperty')

contactOption: *URIRef* = rdflib.term.URIRef('https://schema.org/contactOption')

contactPoint: *URIRef* = rdflib.term.URIRef('https://schema.org/contactPoint')

contactPoints: *URIRef* = rdflib.term.URIRef('https://schema.org/contactPoints')

contactType: *URIRef* = rdflib.term.URIRef('https://schema.org/contactType')

contactlessPayment: *URIRef* = rdflib.term.URIRef('https://schema.org/contactlessPayment')

containedIn: *URIRef* = rdflib.term.URIRef('https://schema.org/containedIn')

containedInPlace: *URIRef* = rdflib.term.URIRef('https://schema.org/containedInPlace')

containsPlace: *URIRef* = rdflib.term.URIRef('https://schema.org/containsPlace')

containsSeason: *URIRef* = rdflib.term.URIRef('https://schema.org/containsSeason')

contentLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/contentLocation')

contentRating: *URIRef* = rdflib.term.URIRef('https://schema.org/contentRating')

contentReferenceTime: *URIRef* = rdflib.term.URIRef('https://schema.org/contentReferenceTime')

contentSize: *URIRef* = rdflib.term.URIRef('https://schema.org/contentSize')

contentType: *URIRef* = rdflib.term.URIRef('https://schema.org/contentType')

contentUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/contentUrl')

contraindication: *URIRef* = rdflib.term.URIRef('https://schema.org/contraindication')

contributor: *URIRef* = rdflib.term.URIRef('https://schema.org/contributor')

cookTime: *URIRef* = rdflib.term.URIRef('https://schema.org/cookTime')

cookingMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/cookingMethod')

copyrightHolder: *URIRef* = rdflib.term.URIRef('https://schema.org/copyrightHolder')

copyrightNotice: *URIRef* = rdflib.term.URIRef('https://schema.org/copyrightNotice')

copyrightYear: *URIRef* = rdflib.term.URIRef('https://schema.org/copyrightYear')

correction: *URIRef* = rdflib.term.URIRef('https://schema.org/correction')

correctionsPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/correctionsPolicy')

costCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/costCategory')

costCurrency: *URIRef* = rdflib.term.URIRef('https://schema.org/costCurrency')

costOrigin: *URIRef* = rdflib.term.URIRef('https://schema.org/costOrigin')

costPerUnit: *URIRef* = rdflib.term.URIRef('https://schema.org/costPerUnit')

countriesNotSupported: *URIRef* = rdflib.term.URIRef('https://schema.org/countriesNotSupported')

countriesSupported: *URIRef* = rdflib.term.URIRef('https://schema.org/countriesSupported')

countryOfAssembly: *URIRef* = rdflib.term.URIRef('https://schema.org/countryOfAssembly')

countryOfLastProcessing: *URIRef* = rdflib.term.URIRef('https://schema.org/countryOfLastProcessing')

countryOfOrigin: *URIRef* = rdflib.term.URIRef('https://schema.org/countryOfOrigin')

course: *URIRef* = rdflib.term.URIRef('https://schema.org/course')

courseCode: *URIRef* = rdflib.term.URIRef('https://schema.org/courseCode')

courseMode: *URIRef* = rdflib.term.URIRef('https://schema.org/courseMode')

coursePrerequisites: *URIRef* =
rdflib.term.URIRef('https://schema.org/coursePrerequisites')

courseWorkload: *URIRef* = rdflib.term.URIRef('https://schema.org/courseWorkload')

coverageEndTime: *URIRef* = rdflib.term.URIRef('https://schema.org/coverageEndTime')

coverageStartTime: *URIRef* =
rdflib.term.URIRef('https://schema.org/coverageStartTime')

creativeWorkStatus: *URIRef* =
rdflib.term.URIRef('https://schema.org/creativeWorkStatus')

creator: *URIRef* = rdflib.term.URIRef('https://schema.org/creator')

credentialCategory: *URIRef* =
rdflib.term.URIRef('https://schema.org/credentialCategory')

creditText: *URIRef* = rdflib.term.URIRef('https://schema.org/creditText')

creditedTo: *URIRef* = rdflib.term.URIRef('https://schema.org/creditedTo')

cssSelector: *URIRef* = rdflib.term.URIRef('https://schema.org/cssSelector')

currenciesAccepted: *URIRef* =
rdflib.term.URIRef('https://schema.org/currenciesAccepted')

currency: *URIRef* = rdflib.term.URIRef('https://schema.org/currency')

currentExchangeRate: *URIRef* =
rdflib.term.URIRef('https://schema.org/currentExchangeRate')

customer: *URIRef* = rdflib.term.URIRef('https://schema.org/customer')

customerRemorseReturnFees: *URIRef* =
rdflib.term.URIRef('https://schema.org/customerRemorseReturnFees')

customerRemorseReturnLabelSource: *URIRef* =
rdflib.term.URIRef('https://schema.org/customerRemorseReturnLabelSource')

customerRemorseReturnShippingFeesAmount: *URIRef* =
rdflib.term.URIRef('https://schema.org/customerRemorseReturnShippingFeesAmount')

cutoffTime: *URIRef* = rdflib.term.URIRef('https://schema.org/cutoffTime')

cvdCollectionDate: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdCollectionDate')

cvdFacilityCounty: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdFacilityCounty')

cvdFacilityId: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdFacilityId')

cvdNumBeds: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumBeds')

cvdNumBedsOcc: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumBedsOcc')

cvdNumC19Died: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumC19Died')

cvdNumC19HOPats: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumC19HOPats')

cvdNumC19HospPats: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdNumC19HospPats')

cvdNumC19MechVentPats: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdNumC19MechVentPats')

cvdNumC19OFMechVentPats: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdNumC19OFMechVentPats')

cvdNumC19OverflowPats: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdNumC19OverflowPats')

cvdNumICUBeds: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumICUBeds')

cvdNumICUBedsOcc: *URIRef* =
rdflib.term.URIRef('https://schema.org/cvdNumICUBedsOcc')

cvdNumTotBeds: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumTotBeds')

cvdNumVent: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumVent')

cvdNumVentUse: *URIRef* = rdflib.term.URIRef('https://schema.org/cvdNumVentUse')

dataFeedElement: *URIRef* = rdflib.term.URIRef('https://schema.org/dataFeedElement')

dataset: *URIRef* = rdflib.term.URIRef('https://schema.org/dataset')

datasetTimeInterval: *URIRef* =
rdflib.term.URIRef('https://schema.org/datasetTimeInterval')

dateCreated: *URIRef* = rdflib.term.URIRef('https://schema.org/dateCreated')

dateDeleted: *URIRef* = rdflib.term.URIRef('https://schema.org/dateDeleted')

dateIssued: *URIRef* = rdflib.term.URIRef('https://schema.org/dateIssued')

dateModified: *URIRef* = rdflib.term.URIRef('https://schema.org/dateModified')

datePosted: *URIRef* = rdflib.term.URIRef('https://schema.org/datePosted')

datePublished: *URIRef* = rdflib.term.URIRef('https://schema.org/datePublished')

dateRead: *URIRef* = rdflib.term.URIRef('https://schema.org/dateRead')

dateReceived: *URIRef* = rdflib.term.URIRef('https://schema.org/dateReceived')

dateSent: *URIRef* = rdflib.term.URIRef('https://schema.org/dateSent')

dateVehicleFirstRegistered: *URIRef* =
rdflib.term.URIRef('https://schema.org/dateVehicleFirstRegistered')

dateline: *URIRef* = rdflib.term.URIRef('https://schema.org/dateline')

dayOfWeek: *URIRef* = rdflib.term.URIRef('https://schema.org/dayOfWeek')

deathDate: *URIRef* = rdflib.term.URIRef('https://schema.org/deathDate')

deathPlace: *URIRef* = rdflib.term.URIRef('https://schema.org/deathPlace')

defaultValue: *URIRef* = rdflib.term.URIRef('https://schema.org/defaultValue')

deliveryAddress: *URIRef* = rdflib.term.URIRef('https://schema.org/deliveryAddress')

deliveryLeadTime: *URIRef* =
rdflib.term.URIRef('https://schema.org/deliveryLeadTime')

deliveryMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/deliveryMethod')

deliveryStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/deliveryStatus')

deliveryTime: *URIRef* = rdflib.term.URIRef('https://schema.org/deliveryTime')

department: *URIRef* = rdflib.term.URIRef('https://schema.org/department')

departureAirport: *URIRef* =
rdflib.term.URIRef('https://schema.org/departureAirport')

departureBoatTerminal: *URIRef* =
rdflib.term.URIRef('https://schema.org/departureBoatTerminal')

departureBusStop: *URIRef* =
rdflib.term.URIRef('https://schema.org/departureBusStop')

departureGate: *URIRef* = rdflib.term.URIRef('https://schema.org/departureGate')

departurePlatform: *URIRef* =
rdflib.term.URIRef('https://schema.org/departurePlatform')

departureStation: *URIRef* =
rdflib.term.URIRef('https://schema.org/departureStation')

departureTerminal: *URIRef* =
rdflib.term.URIRef('https://schema.org/departureTerminal')

departureTime: *URIRef* = rdflib.term.URIRef('https://schema.org/departureTime')

dependencies: *URIRef* = rdflib.term.URIRef('https://schema.org/dependencies')

depth: *URIRef* = rdflib.term.URIRef('https://schema.org/depth')

description: *URIRef* = rdflib.term.URIRef('https://schema.org/description')

device: *URIRef* = rdflib.term.URIRef('https://schema.org/device')

diagnosis: *URIRef* = rdflib.term.URIRef('https://schema.org/diagnosis')

diagram: *URIRef* = rdflib.term.URIRef('https://schema.org/diagram')

diet: *URIRef* = rdflib.term.URIRef('https://schema.org/diet')

dietFeatures: *URIRef* = rdflib.term.URIRef('https://schema.org/dietFeatures')

differentialDiagnosis: *URIRef* =
rdflib.term.URIRef('https://schema.org/differentialDiagnosis')

directApply: *URIRef* = rdflib.term.URIRef('https://schema.org/directApply')

director: *URIRef* = rdflib.term.URIRef('https://schema.org/director')

directors: *URIRef* = rdflib.term.URIRef('https://schema.org/directors')

disambiguatingDescription: *URIRef* =
rdflib.term.URIRef('https://schema.org/disambiguatingDescription')

discount: *URIRef* = rdflib.term.URIRef('https://schema.org/discount')

discountCode: *URIRef* = rdflib.term.URIRef('https://schema.org/discountCode')

discountCurrency: *URIRef* =
rdflib.term.URIRef('https://schema.org/discountCurrency')

discusses: *URIRef* = rdflib.term.URIRef('https://schema.org/discusses')

discussionUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/discussionUrl')

diseasePreventionInfo: *URIRef* =
rdflib.term.URIRef('https://schema.org/diseasePreventionInfo')

diseaseSpreadStatistics: *URIRef* =
rdflib.term.URIRef('https://schema.org/diseaseSpreadStatistics')

dissolutionDate: *URIRef* = rdflib.term.URIRef('https://schema.org/dissolutionDate')

distance: *URIRef* = rdflib.term.URIRef('https://schema.org/distance')

distinguishingSign: *URIRef* =
rdflib.term.URIRef('https://schema.org/distinguishingSign')

distribution: *URIRef* = rdflib.term.URIRef('https://schema.org/distribution')

diversityPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/diversityPolicy')

diversityStaffingReport: *URIRef* =
rdflib.term.URIRef('https://schema.org/diversityStaffingReport')

documentation: *URIRef* = rdflib.term.URIRef('https://schema.org/documentation')

doesNotShip: *URIRef* = rdflib.term.URIRef('https://schema.org/doesNotShip')

domainIncludes: *URIRef* = rdflib.term.URIRef('https://schema.org/domainIncludes')

domiciledMortgage: *URIRef* =
rdflib.term.URIRef('https://schema.org/domiciledMortgage')

doorTime: *URIRef* = rdflib.term.URIRef('https://schema.org/doorTime')

dosageForm: *URIRef* = rdflib.term.URIRef('https://schema.org/dosageForm')

doseSchedule: *URIRef* = rdflib.term.URIRef('https://schema.org/doseSchedule')

doseUnit: *URIRef* = rdflib.term.URIRef('https://schema.org/doseUnit')

doseValue: *URIRef* = rdflib.term.URIRef('https://schema.org/doseValue')

downPayment: *URIRef* = rdflib.term.URIRef('https://schema.org/downPayment')

downloadUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/downloadUrl')

downvoteCount: *URIRef* = rdflib.term.URIRef('https://schema.org/downvoteCount')

drainsTo: *URIRef* = rdflib.term.URIRef('https://schema.org/drainsTo')

driveWheelConfiguration: *URIRef* =
rdflib.term.URIRef('https://schema.org/driveWheelConfiguration')

dropoffLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/dropoffLocation')

dropoffTime: *URIRef* = rdflib.term.URIRef('https://schema.org/dropoffTime')

drug: *URIRef* = rdflib.term.URIRef('https://schema.org/drug')

drugClass: *URIRef* = rdflib.term.URIRef('https://schema.org/drugClass')

drugUnit: *URIRef* = rdflib.term.URIRef('https://schema.org/drugUnit')

duns: *URIRef* = rdflib.term.URIRef('https://schema.org/duns')

duplicateTherapy: *URIRef* =
rdflib.term.URIRef('https://schema.org/duplicateTherapy')

duration: *URIRef* = rdflib.term.URIRef('https://schema.org/duration')

durationOfWarranty: *URIRef* =
rdflib.term.URIRef('https://schema.org/durationOfWarranty')

duringMedia: *URIRef* = rdflib.term.URIRef('https://schema.org/duringMedia')

earlyPrepaymentPenalty: *URIRef* =
rdflib.term.URIRef('https://schema.org/earlyPrepaymentPenalty')

editEIDR: *URIRef* = rdflib.term.URIRef('https://schema.org/editEIDR')

editor: *URIRef* = rdflib.term.URIRef('https://schema.org/editor')

eduQuestionType: *URIRef* = rdflib.term.URIRef('https://schema.org/eduQuestionType')

educationRequirements: *URIRef* =
rdflib.term.URIRef('https://schema.org/educationRequirements')

educationalAlignment: *URIRef* =
rdflib.term.URIRef('https://schema.org/educationalAlignment')

educationalCredentialAwarded: *URIRef* =
rdflib.term.URIRef('https://schema.org/educationalCredentialAwarded')

educationalFramework: *URIRef* =
rdflib.term.URIRef('https://schema.org/educationalFramework')

educationalLevel: *URIRef* =
rdflib.term.URIRef('https://schema.org/educationalLevel')

educationalProgramMode: *URIRef* =
rdflib.term.URIRef('https://schema.org/educationalProgramMode')

educationalRole: *URIRef* = rdflib.term.URIRef('https://schema.org/educationalRole')

educationalUse: *URIRef* = rdflib.term.URIRef('https://schema.org/educationalUse')

elevation: *URIRef* = rdflib.term.URIRef('https://schema.org/elevation')

eligibilityToWorkRequirement: *URIRef* =
rdflib.term.URIRef('https://schema.org/eligibilityToWorkRequirement')

eligibleCustomerType: *URIRef* =
rdflib.term.URIRef('https://schema.org/eligibleCustomerType')

eligibleDuration: *URIRef* =
rdflib.term.URIRef('https://schema.org/eligibleDuration')

eligibleQuantity: *URIRef* =
rdflib.term.URIRef('https://schema.org/eligibleQuantity')

eligibleRegion: *URIRef* = rdflib.term.URIRef('https://schema.org/eligibleRegion')

eligibleTransactionVolume: *URIRef* =
rdflib.term.URIRef('https://schema.org/eligibleTransactionVolume')

email: *URIRef* = rdflib.term.URIRef('https://schema.org/email')

embedUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/embedUrl')

embeddedTextCaption: *URIRef* =
rdflib.term.URIRef('https://schema.org/embeddedTextCaption')

emissionsCO2: *URIRef* = rdflib.term.URIRef('https://schema.org/emissionsCO2')

employee: *URIRef* = rdflib.term.URIRef('https://schema.org/employee')

employees: *URIRef* = rdflib.term.URIRef('https://schema.org/employees')

employerOverview: *URIRef* =
rdflib.term.URIRef('https://schema.org/employerOverview')

employmentType: *URIRef* = rdflib.term.URIRef('https://schema.org/employmentType')

employmentUnit: *URIRef* = rdflib.term.URIRef('https://schema.org/employmentUnit')

encodesBioChemEntity: *URIRef* =
rdflib.term.URIRef('https://schema.org/encodesBioChemEntity')

encodesCreativeWork: *URIRef* =
rdflib.term.URIRef('https://schema.org/encodesCreativeWork')

encoding: *URIRef* = rdflib.term.URIRef('https://schema.org/encoding')

encodingFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/encodingFormat')

encodingType: *URIRef* = rdflib.term.URIRef('https://schema.org/encodingType')

encodings: *URIRef* = rdflib.term.URIRef('https://schema.org/encodings')

endDate: *URIRef* = rdflib.term.URIRef('https://schema.org/endDate')

endOffset: *URIRef* = rdflib.term.URIRef('https://schema.org/endOffset')

endTime: *URIRef* = rdflib.term.URIRef('https://schema.org/endTime')

endorsee: *URIRef* = rdflib.term.URIRef('https://schema.org/endorsee')

endorsers: *URIRef* = rdflib.term.URIRef('https://schema.org/endorsers')

energyEfficiencyScaleMax: *URIRef* =
rdflib.term.URIRef('https://schema.org/energyEfficiencyScaleMax')

energyEfficiencyScaleMin: *URIRef* =
rdflib.term.URIRef('https://schema.org/energyEfficiencyScaleMin')

engineDisplacement: *URIRef* =
rdflib.term.URIRef('https://schema.org/engineDisplacement')

enginePower: *URIRef* = rdflib.term.URIRef('https://schema.org/enginePower')

engineType: *URIRef* = rdflib.term.URIRef('https://schema.org/engineType')

entertainmentBusiness: *URIRef* =
rdflib.term.URIRef('https://schema.org/entertainmentBusiness')

epidemiology: *URIRef* = rdflib.term.URIRef('https://schema.org/epidemiology')

episode: *URIRef* = rdflib.term.URIRef('https://schema.org/episode')

episodeNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/episodeNumber')

episodes: *URIRef* = rdflib.term.URIRef('https://schema.org/episodes')

equal: *URIRef* = rdflib.term.URIRef('https://schema.org/equal')

error: *URIRef* = rdflib.term.URIRef('https://schema.org/error')

estimatedCost: *URIRef* = rdflib.term.URIRef('https://schema.org/estimatedCost')

estimatedFlightDuration: *URIRef* =
rdflib.term.URIRef('https://schema.org/estimatedFlightDuration')

estimatedSalary: *URIRef* = rdflib.term.URIRef('https://schema.org/estimatedSalary')

estimatesRiskOf: *URIRef* = rdflib.term.URIRef('https://schema.org/estimatesRiskOf')

ethicsPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/ethicsPolicy')

event: *URIRef* = rdflib.term.URIRef('https://schema.org/event')

eventAttendanceMode: *URIRef* =
rdflib.term.URIRef('https://schema.org/eventAttendanceMode')

eventSchedule: *URIRef* = rdflib.term.URIRef('https://schema.org/eventSchedule')

eventStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/eventStatus')

events: *URIRef* = rdflib.term.URIRef('https://schema.org/events')

evidenceLevel: *URIRef* = rdflib.term.URIRef('https://schema.org/evidenceLevel')

evidenceOrigin: *URIRef* = rdflib.term.URIRef('https://schema.org/evidenceOrigin')

exampleOfWork: *URIRef* = rdflib.term.URIRef('https://schema.org/exampleOfWork')

exceptDate: *URIRef* = rdflib.term.URIRef('https://schema.org/exceptDate')

exchangeRateSpread: *URIRef* =
rdflib.term.URIRef('https://schema.org/exchangeRateSpread')

executableLibraryName: *URIRef* =
rdflib.term.URIRef('https://schema.org/executableLibraryName')

exerciseCourse: *URIRef* = rdflib.term.URIRef('https://schema.org/exerciseCourse')

exercisePlan: *URIRef* = rdflib.term.URIRef('https://schema.org/exercisePlan')

exerciseRelatedDiet: *URIRef* =
rdflib.term.URIRef('https://schema.org/exerciseRelatedDiet')

exerciseType: *URIRef* = rdflib.term.URIRef('https://schema.org/exerciseType')

exifData: *URIRef* = rdflib.term.URIRef('https://schema.org/exifData')

expectedArrivalFrom: *URIRef* =
rdflib.term.URIRef('https://schema.org/expectedArrivalFrom')

expectedArrivalUntil: *URIRef* =
rdflib.term.URIRef('https://schema.org/expectedArrivalUntil')

expectedPrognosis: *URIRef* =
rdflib.term.URIRef('https://schema.org/expectedPrognosis')

expectsAcceptanceOf: *URIRef* =
rdflib.term.URIRef('https://schema.org/expectsAcceptanceOf')

experienceInPlaceOfEducation: *URIRef* =
rdflib.term.URIRef('https://schema.org/experienceInPlaceOfEducation')

experienceRequirements: *URIRef* =
rdflib.term.URIRef('https://schema.org/experienceRequirements')

expertConsiderations: *URIRef* =
rdflib.term.URIRef('https://schema.org/expertConsiderations')

expires: *URIRef* = rdflib.term.URIRef('https://schema.org/expires')

expressedIn: *URIRef* = rdflib.term.URIRef('https://schema.org/expressedIn')

familyName: *URIRef* = rdflib.term.URIRef('https://schema.org/familyName')

fatContent: *URIRef* = rdflib.term.URIRef('https://schema.org/fatContent')

faxNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/faxNumber')

featureList: *URIRef* = rdflib.term.URIRef('https://schema.org/featureList')

feesAndCommissionsSpecification: *URIRef* =
rdflib.term.URIRef('https://schema.org/feesAndCommissionsSpecification')

fiberContent: *URIRef* = rdflib.term.URIRef('https://schema.org/fiberContent')

fileFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/fileFormat')

fileSize: *URIRef* = rdflib.term.URIRef('https://schema.org/fileSize')

financialAidEligible: *URIRef* = rdflib.term.URIRef('https://schema.org/financialAidEligible')

firstAppearance: *URIRef* = rdflib.term.URIRef('https://schema.org/firstAppearance')

firstPerformance: *URIRef* = rdflib.term.URIRef('https://schema.org/firstPerformance')

flightDistance: *URIRef* = rdflib.term.URIRef('https://schema.org/flightDistance')

flightNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/flightNumber')

floorLevel: *URIRef* = rdflib.term.URIRef('https://schema.org/floorLevel')

floorLimit: *URIRef* = rdflib.term.URIRef('https://schema.org/floorLimit')

floorSize: *URIRef* = rdflib.term.URIRef('https://schema.org/floorSize')

followee: *URIRef* = rdflib.term.URIRef('https://schema.org/followee')

follows: *URIRef* = rdflib.term.URIRef('https://schema.org/follows')

followup: *URIRef* = rdflib.term.URIRef('https://schema.org/followup')

foodEstablishment: *URIRef* = rdflib.term.URIRef('https://schema.org/foodEstablishment')

foodEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/foodEvent')

foodWarning: *URIRef* = rdflib.term.URIRef('https://schema.org/foodWarning')

founder: *URIRef* = rdflib.term.URIRef('https://schema.org/founder')

founders: *URIRef* = rdflib.term.URIRef('https://schema.org/founders')

foundingDate: *URIRef* = rdflib.term.URIRef('https://schema.org/foundingDate')

foundingLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/foundingLocation')

free: *URIRef* = rdflib.term.URIRef('https://schema.org/free')

freeShippingThreshold: *URIRef* = rdflib.term.URIRef('https://schema.org/freeShippingThreshold')

frequency: *URIRef* = rdflib.term.URIRef('https://schema.org/frequency')

fromLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/fromLocation')

fuelCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/fuelCapacity')

fuelConsumption: *URIRef* = rdflib.term.URIRef('https://schema.org/fuelConsumption')

fuelEfficiency: *URIRef* = rdflib.term.URIRef('https://schema.org/fuelEfficiency')

fuelType: *URIRef* = rdflib.term.URIRef('https://schema.org/fuelType')

functionalClass: *URIRef* = rdflib.term.URIRef('https://schema.org/functionalClass')

fundedItem: *URIRef* = rdflib.term.URIRef('https://schema.org/fundedItem')

funder: *URIRef* = rdflib.term.URIRef('https://schema.org/funder')

game: *URIRef* = rdflib.term.URIRef('https://schema.org/game')

gameItem: *URIRef* = rdflib.term.URIRef('https://schema.org/gameItem')

gameLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/gameLocation')

gamePlatform: *URIRef* = rdflib.term.URIRef('https://schema.org/gamePlatform')

gameServer: *URIRef* = rdflib.term.URIRef('https://schema.org/gameServer')

gameTip: *URIRef* = rdflib.term.URIRef('https://schema.org/gameTip')

gender: *URIRef* = rdflib.term.URIRef('https://schema.org/gender')

genre: *URIRef* = rdflib.term.URIRef('https://schema.org/genre')

geo: *URIRef* = rdflib.term.URIRef('https://schema.org/geo')

geoContains: *URIRef* = rdflib.term.URIRef('https://schema.org/geoContains')

geoCoveredBy: *URIRef* = rdflib.term.URIRef('https://schema.org/geoCoveredBy')

geoCovers: *URIRef* = rdflib.term.URIRef('https://schema.org/geoCovers')

geoCrosses: *URIRef* = rdflib.term.URIRef('https://schema.org/geoCrosses')

geoDisjoint: *URIRef* = rdflib.term.URIRef('https://schema.org/geoDisjoint')

geoEquals: *URIRef* = rdflib.term.URIRef('https://schema.org/geoEquals')

geoIntersects: *URIRef* = rdflib.term.URIRef('https://schema.org/geoIntersects')

geoMidpoint: *URIRef* = rdflib.term.URIRef('https://schema.org/geoMidpoint')

geoOverlaps: *URIRef* = rdflib.term.URIRef('https://schema.org/geoOverlaps')

geoRadius: *URIRef* = rdflib.term.URIRef('https://schema.org/geoRadius')

geoTouches: *URIRef* = rdflib.term.URIRef('https://schema.org/geoTouches')

geoWithin: *URIRef* = rdflib.term.URIRef('https://schema.org/geoWithin')

geographicArea: *URIRef* = rdflib.term.URIRef('https://schema.org/geographicArea')

gettingTestedInfo: *URIRef* =
rdflib.term.URIRef('https://schema.org/gettingTestedInfo')

givenName: *URIRef* = rdflib.term.URIRef('https://schema.org/givenName')

globalLocationNumber: *URIRef* =
rdflib.term.URIRef('https://schema.org/globalLocationNumber')

governmentBenefitsInfo: *URIRef* = rdflib.term.URIRef('https://schema.org/governmentBenefitsInfo')

gracePeriod: *URIRef* = rdflib.term.URIRef('https://schema.org/gracePeriod')

grantee: *URIRef* = rdflib.term.URIRef('https://schema.org/grantee')

greater: *URIRef* = rdflib.term.URIRef('https://schema.org/greater')

greaterOrEqual: *URIRef* = rdflib.term.URIRef('https://schema.org/greaterOrEqual')

gtin: *URIRef* = rdflib.term.URIRef('https://schema.org/gtin')

gtin12: *URIRef* = rdflib.term.URIRef('https://schema.org/gtin12')

gtin13: *URIRef* = rdflib.term.URIRef('https://schema.org/gtin13')

gtin14: *URIRef* = rdflib.term.URIRef('https://schema.org/gtin14')

gtin8: *URIRef* = rdflib.term.URIRef('https://schema.org/gtin8')

guideline: *URIRef* = rdflib.term.URIRef('https://schema.org/guideline')

guidelineDate: *URIRef* = rdflib.term.URIRef('https://schema.org/guidelineDate')

guidelineSubject: *URIRef* = rdflib.term.URIRef('https://schema.org/guidelineSubject')

handlingTime: *URIRef* = rdflib.term.URIRef('https://schema.org/handlingTime')

hasBioChemEntityPart: *URIRef* = rdflib.term.URIRef('https://schema.org/hasBioChemEntityPart')

hasBioPolymerSequence: *URIRef* = rdflib.term.URIRef('https://schema.org/hasBioPolymerSequence')

hasBroadcastChannel: *URIRef* = rdflib.term.URIRef('https://schema.org/hasBroadcastChannel')

hasCategoryCode: *URIRef* = rdflib.term.URIRef('https://schema.org/hasCategoryCode')

hasCourse: *URIRef* = rdflib.term.URIRef('https://schema.org/hasCourse')

hasCourseInstance: *URIRef* = rdflib.term.URIRef('https://schema.org/hasCourseInstance')

hasCredential: *URIRef* = rdflib.term.URIRef('https://schema.org/hasCredential')

hasDefinedTerm: *URIRef* = rdflib.term.URIRef('https://schema.org/hasDefinedTerm')

hasDeliveryMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/hasDeliveryMethod')

hasDigitalDocumentPermission: *URIRef* = rdflib.term.URIRef('https://schema.org/hasDigitalDocumentPermission')

hasDriveThroughService: *URIRef* = rdflib.term.URIRef('https://schema.org/hasDriveThroughService')

hasEnergyConsumptionDetails: *URIRef* = rdflib.term.URIRef('https://schema.org/hasEnergyConsumptionDetails')

hasEnergyEfficiencyCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/hasEnergyEfficiencyCategory')

hasHealthAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/hasHealthAspect')

hasMap: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMap')

hasMeasurement: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMeasurement')

hasMenu: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMenu')

hasMenuItem: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMenuItem')

hasMenuSection: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMenuSection')

hasMerchantReturnPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMerchantReturnPolicy')

hasMolecularFunction: *URIRef* = rdflib.term.URIRef('https://schema.org/hasMolecularFunction')

hasOccupation: *URIRef* = rdflib.term.URIRef('https://schema.org/hasOccupation')

hasOfferCatalog: *URIRef* = rdflib.term.URIRef('https://schema.org/hasOfferCatalog')

hasPOS: *URIRef* = rdflib.term.URIRef('https://schema.org/hasPOS')

hasPart: *URIRef* = rdflib.term.URIRef('https://schema.org/hasPart')

hasRepresentation: *URIRef* = rdflib.term.URIRef('https://schema.org/hasRepresentation')

hasVariant: *URIRef* = rdflib.term.URIRef('https://schema.org/hasVariant')

headline: *URIRef* = rdflib.term.URIRef('https://schema.org/headline')

healthCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/healthCondition')

healthPlanCoinsuranceOption: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanCoinsuranceOption')

healthPlanCoinsuranceRate: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanCoinsuranceRate')

healthPlanCopay: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanCopay')

healthPlanCopayOption: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanCopayOption')

healthPlanCostSharing: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanCostSharing')

healthPlanDrugOption: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanDrugOption')

healthPlanDrugTier: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanDrugTier')

healthPlanId: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanId')

healthPlanMarketingUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanMarketingUrl')

healthPlanNetworkId: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanNetworkId')

healthPlanNetworkTier: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanNetworkTier')

healthPlanPharmacyCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/healthPlanPharmacyCategory')

healthcareReportingData: *URIRef* = rdflib.term.URIRef('https://schema.org/healthcareReportingData')

height: *URIRef* = rdflib.term.URIRef('https://schema.org/height')

highPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/highPrice')

hiringOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/hiringOrganization')

holdingArchive: *URIRef* = rdflib.term.URIRef('https://schema.org/holdingArchive')

homeLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/homeLocation')

homeTeam: *URIRef* = rdflib.term.URIRef('https://schema.org/homeTeam')

honorificPrefix: *URIRef* = rdflib.term.URIRef('https://schema.org/honorificPrefix')

honorificSuffix: *URIRef* = rdflib.term.URIRef('https://schema.org/honorificSuffix')

hospitalAffiliation: *URIRef* = rdflib.term.URIRef('https://schema.org/hospitalAffiliation')

hostingOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/hostingOrganization')

hoursAvailable: *URIRef* = rdflib.term.URIRef('https://schema.org/hoursAvailable')

howPerformed: *URIRef* = rdflib.term.URIRef('https://schema.org/howPerformed')

httpMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/httpMethod')

iataCode: *URIRef* = rdflib.term.URIRef('https://schema.org/iataCode')

icaoCode: *URIRef* = rdflib.term.URIRef('https://schema.org/icaoCode')

identifier: *URIRef* = rdflib.term.URIRef('https://schema.org/identifier')

identifyingExam: *URIRef* = rdflib.term.URIRef('https://schema.org/identifyingExam')

identifyingTest: *URIRef* = rdflib.term.URIRef('https://schema.org/identifyingTest')

illustrator: *URIRef* = rdflib.term.URIRef('https://schema.org/illustrator')

image: *URIRef* = rdflib.term.URIRef('https://schema.org/image')

imagingTechnique: *URIRef* =
rdflib.term.URIRef('https://schema.org/imagingTechnique')

inAlbum: *URIRef* = rdflib.term.URIRef('https://schema.org/inAlbum')

inBroadcastLineup: *URIRef* =
rdflib.term.URIRef('https://schema.org/inBroadcastLineup')

inChI: *URIRef* = rdflib.term.URIRef('https://schema.org/inChI')

inChIKey: *URIRef* = rdflib.term.URIRef('https://schema.org/inChIKey')

inCodeSet: *URIRef* = rdflib.term.URIRef('https://schema.org/inCodeSet')

inDefinedTermSet: *URIRef* =
rdflib.term.URIRef('https://schema.org/inDefinedTermSet')

inLanguage: *URIRef* = rdflib.term.URIRef('https://schema.org/inLanguage')

inPlaylist: *URIRef* = rdflib.term.URIRef('https://schema.org/inPlaylist')

inProductGroupWithID: *URIRef* =
rdflib.term.URIRef('https://schema.org/inProductGroupWithID')

inStoreReturnsOffered: *URIRef* =
rdflib.term.URIRef('https://schema.org/inStoreReturnsOffered')

inSupportOf: *URIRef* = rdflib.term.URIRef('https://schema.org/inSupportOf')

incentiveCompensation: *URIRef* =
rdflib.term.URIRef('https://schema.org/incentiveCompensation')

incentives: *URIRef* = rdflib.term.URIRef('https://schema.org/incentives')

includedComposition: *URIRef* =
rdflib.term.URIRef('https://schema.org/includedComposition')

includedDataCatalog: *URIRef* =
rdflib.term.URIRef('https://schema.org/includedDataCatalog')

includedInDataCatalog: *URIRef* =
rdflib.term.URIRef('https://schema.org/includedInDataCatalog')

includedInHealthInsurancePlan: *URIRef* =
rdflib.term.URIRef('https://schema.org/includedInHealthInsurancePlan')

includedRiskFactor: *URIRef* =
rdflib.term.URIRef('https://schema.org/includedRiskFactor')

includesAttraction: *URIRef* =
rdflib.term.URIRef('https://schema.org/includesAttraction')

includesHealthPlanFormulary: *URIRef* =
rdflib.term.URIRef('https://schema.org/includesHealthPlanFormulary')

includesHealthPlanNetwork: *URIRef* = rdflib.term.URIRef('https://schema.org/includesHealthPlanNetwork')

includesObject: *URIRef* = rdflib.term.URIRef('https://schema.org/includesObject')

increasesRiskOf: *URIRef* = rdflib.term.URIRef('https://schema.org/increasesRiskOf')

industry: *URIRef* = rdflib.term.URIRef('https://schema.org/industry')

ineligibleRegion: *URIRef* = rdflib.term.URIRef('https://schema.org/ineligibleRegion')

infectiousAgent: *URIRef* = rdflib.term.URIRef('https://schema.org/infectiousAgent')

infectiousAgentClass: *URIRef* = rdflib.term.URIRef('https://schema.org/infectiousAgentClass')

ingredients: *URIRef* = rdflib.term.URIRef('https://schema.org/ingredients')

inker: *URIRef* = rdflib.term.URIRef('https://schema.org/inker')

insertion: *URIRef* = rdflib.term.URIRef('https://schema.org/insertion')

installUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/installUrl')

instructor: *URIRef* = rdflib.term.URIRef('https://schema.org/instructor')

instrument: *URIRef* = rdflib.term.URIRef('https://schema.org/instrument')

intensity: *URIRef* = rdflib.term.URIRef('https://schema.org/intensity')

interactingDrug: *URIRef* = rdflib.term.URIRef('https://schema.org/interactingDrug')

interactionCount: *URIRef* = rdflib.term.URIRef('https://schema.org/interactionCount')

interactionService: *URIRef* = rdflib.term.URIRef('https://schema.org/interactionService')

interactionStatistic: *URIRef* = rdflib.term.URIRef('https://schema.org/interactionStatistic')

interactionType: *URIRef* = rdflib.term.URIRef('https://schema.org/interactionType')

interactivityType: *URIRef* = rdflib.term.URIRef('https://schema.org/interactivityType')

interestRate: *URIRef* = rdflib.term.URIRef('https://schema.org/interestRate')

interpretedAsClaim: *URIRef* = rdflib.term.URIRef('https://schema.org/interpretedAsClaim')

inventoryLevel: *URIRef* = rdflib.term.URIRef('https://schema.org/inventoryLevel')

inverseOf: *URIRef* = rdflib.term.URIRef('https://schema.org/inverseOf')

isAcceptingNewPatients: *URIRef* = rdflib.term.URIRef('https://schema.org/isAcceptingNewPatients')

isAccessibleForFree: *URIRef* = rdflib.term.URIRef('https://schema.org/isAccessibleForFree')

isAccessoryOrSparePartFor: *URIRef* = rdflib.term.URIRef('https://schema.org/isAccessoryOrSparePartFor')

isAvailableGenerically: *URIRef* = rdflib.term.URIRef('https://schema.org/isAvailableGenerically')

isBasedOn: *URIRef* = rdflib.term.URIRef('https://schema.org/isBasedOn')

isBasedOnUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/isBasedOnUrl')

isConsumableFor: *URIRef* = rdflib.term.URIRef('https://schema.org/isConsumableFor')

isEncodedByBioChemEntity: *URIRef* = rdflib.term.URIRef('https://schema.org/isEncodedByBioChemEntity')

isFamilyFriendly: *URIRef* = rdflib.term.URIRef('https://schema.org/isFamilyFriendly')

isGift: *URIRef* = rdflib.term.URIRef('https://schema.org/isGift')

isInvolvedInBiologicalProcess: *URIRef* = rdflib.term.URIRef('https://schema.org/isInvolvedInBiologicalProcess')

isLiveBroadcast: *URIRef* = rdflib.term.URIRef('https://schema.org/isLiveBroadcast')

isLocatedInSubcellularLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/isLocatedInSubcellularLocation')

isPartOf: *URIRef* = rdflib.term.URIRef('https://schema.org/isPartOf')

isPartOfBioChemEntity: *URIRef* = rdflib.term.URIRef('https://schema.org/isPartOfBioChemEntity')

isPlanForApartment: *URIRef* = rdflib.term.URIRef('https://schema.org/isPlanForApartment')

isProprietary: *URIRef* = rdflib.term.URIRef('https://schema.org/isProprietary')

isRelatedTo: *URIRef* = rdflib.term.URIRef('https://schema.org/isRelatedTo')

isResizable: *URIRef* = rdflib.term.URIRef('https://schema.org/isResizable')

isSimilarTo: *URIRef* = rdflib.term.URIRef('https://schema.org/isSimilarTo')

isUnlabelledFallback: *URIRef* = rdflib.term.URIRef('https://schema.org/isUnlabelledFallback')

isVariantOf: *URIRef* = rdflib.term.URIRef('https://schema.org/isVariantOf')

isbn: *URIRef* = rdflib.term.URIRef('https://schema.org/isbn')

isicV4: *URIRef* = rdflib.term.URIRef('https://schema.org/isicV4')

isrcCode: *URIRef* = rdflib.term.URIRef('https://schema.org/isrcCode')

issn: *URIRef* = rdflib.term.URIRef('https://schema.org/issn')

issueNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/issueNumber')

issuedBy: *URIRef* = rdflib.term.URIRef('https://schema.org/issuedBy')

issuedThrough: *URIRef* = rdflib.term.URIRef('https://schema.org/issuedThrough')

iswcCode: *URIRef* = rdflib.term.URIRef('https://schema.org/iswcCode')

item: *URIRef* = rdflib.term.URIRef('https://schema.org/item')

itemCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/itemCondition')

itemDefectReturnFees: *URIRef* = rdflib.term.URIRef('https://schema.org/itemDefectReturnFees')

itemDefectReturnLabelSource: *URIRef* = rdflib.term.URIRef('https://schema.org/itemDefectReturnLabelSource')

itemDefectReturnShippingFeesAmount: *URIRef* = rdflib.term.URIRef('https://schema.org/itemDefectReturnShippingFeesAmount')

itemListElement: *URIRef* = rdflib.term.URIRef('https://schema.org/itemListElement')

itemListOrder: *URIRef* = rdflib.term.URIRef('https://schema.org/itemListOrder')

itemLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/itemLocation')

itemOffered: *URIRef* = rdflib.term.URIRef('https://schema.org/itemOffered')

itemReviewed: *URIRef* = rdflib.term.URIRef('https://schema.org/itemReviewed')

itemShipped: *URIRef* = rdflib.term.URIRef('https://schema.org/itemShipped')

itinerary: *URIRef* = rdflib.term.URIRef('https://schema.org/itinerary')

iupacName: *URIRef* = rdflib.term.URIRef('https://schema.org/iupacName')

jobBenefits: *URIRef* = rdflib.term.URIRef('https://schema.org/jobBenefits')

jobImmediateStart: *URIRef* = rdflib.term.URIRef('https://schema.org/jobImmediateStart')

jobLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/jobLocation')

jobLocationType: *URIRef* = rdflib.term.URIRef('https://schema.org/jobLocationType')

jobStartDate: *URIRef* = rdflib.term.URIRef('https://schema.org/jobStartDate')

jobTitle: *URIRef* = rdflib.term.URIRef('https://schema.org/jobTitle')

jurisdiction: *URIRef* = rdflib.term.URIRef('https://schema.org/jurisdiction')

keywords: *URIRef* = rdflib.term.URIRef('https://schema.org/keywords')

knownVehicleDamages: *URIRef* = rdflib.term.URIRef('https://schema.org/knownVehicleDamages')

knows: *URIRef* = rdflib.term.URIRef('https://schema.org/knows')

knowsAbout: *URIRef* = rdflib.term.URIRef('https://schema.org/knowsAbout')

knowsLanguage: *URIRef* = rdflib.term.URIRef('https://schema.org/knowsLanguage')

labelDetails: *URIRef* = rdflib.term.URIRef('https://schema.org/labelDetails')

landlord: *URIRef* = rdflib.term.URIRef('https://schema.org/landlord')

language: *URIRef* = rdflib.term.URIRef('https://schema.org/language')

lastReviewed: *URIRef* = rdflib.term.URIRef('https://schema.org/lastReviewed')

latitude: *URIRef* = rdflib.term.URIRef('https://schema.org/latitude')

layoutImage: *URIRef* = rdflib.term.URIRef('https://schema.org/layoutImage')

learningResourceType: *URIRef* =
rdflib.term.URIRef('https://schema.org/learningResourceType')

leaseLength: *URIRef* = rdflib.term.URIRef('https://schema.org/leaseLength')

legalName: *URIRef* = rdflib.term.URIRef('https://schema.org/legalName')

legalStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/legalStatus')

legislationApplies: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationApplies')

legislationChanges: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationChanges')

legislationConsolidates: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationConsolidates')

legislationDate: *URIRef* = rdflib.term.URIRef('https://schema.org/legislationDate')

legislationDateVersion: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationDateVersion')

legislationIdentifier: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationIdentifier')

legislationJurisdiction: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationJurisdiction')

legislationLegalForce: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationLegalForce')

legislationLegalValue: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationLegalValue')

legislationPassedBy: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationPassedBy')

legislationResponsible: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationResponsible')

legislationTransposes: *URIRef* =
rdflib.term.URIRef('https://schema.org/legislationTransposes')

legislationType: *URIRef* = rdflib.term.URIRef('https://schema.org/legislationType')

leiCode: *URIRef* = rdflib.term.URIRef('https://schema.org/leiCode')

lender: *URIRef* = rdflib.term.URIRef('https://schema.org/lender')

lesser: *URIRef* = rdflib.term.URIRef('https://schema.org/lesser')

lesserOrEqual: *URIRef* = rdflib.term.URIRef('https://schema.org/lesserOrEqual')

letterer: *URIRef* = rdflib.term.URIRef('https://schema.org/letterer')

license: *URIRef* = rdflib.term.URIRef('https://schema.org/license')

line: *URIRef* = rdflib.term.URIRef('https://schema.org/line')

linkRelationship: *URIRef* =
rdflib.term.URIRef('https://schema.org/linkRelationship')

liveBlogUpdate: *URIRef* = rdflib.term.URIRef('https://schema.org/liveBlogUpdate')

loanMortgageMandateAmount: *URIRef* =
rdflib.term.URIRef('https://schema.org/loanMortgageMandateAmount')

loanPaymentAmount: *URIRef* =
rdflib.term.URIRef('https://schema.org/loanPaymentAmount')

loanPaymentFrequency: *URIRef* =
rdflib.term.URIRef('https://schema.org/loanPaymentFrequency')

loanRepaymentForm: *URIRef* =
rdflib.term.URIRef('https://schema.org/loanRepaymentForm')

loanTerm: *URIRef* = rdflib.term.URIRef('https://schema.org/loanTerm')

loanType: *URIRef* = rdflib.term.URIRef('https://schema.org/loanType')

location: *URIRef* = rdflib.term.URIRef('https://schema.org/location')

locationCreated: *URIRef* = rdflib.term.URIRef('https://schema.org/locationCreated')

lodgingUnitDescription: *URIRef* =
rdflib.term.URIRef('https://schema.org/lodgingUnitDescription')

lodgingUnitType: *URIRef* = rdflib.term.URIRef('https://schema.org/lodgingUnitType')

logo: *URIRef* = rdflib.term.URIRef('https://schema.org/logo')

longitude: *URIRef* = rdflib.term.URIRef('https://schema.org/longitude')

loser: *URIRef* = rdflib.term.URIRef('https://schema.org/loser')

lowPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/lowPrice')

lyricist: *URIRef* = rdflib.term.URIRef('https://schema.org/lyricist')

lyrics: *URIRef* = rdflib.term.URIRef('https://schema.org/lyrics')

mainContentOfPage: *URIRef* = rdflib.term.URIRef('https://schema.org/mainContentOfPage')

mainEntity: *URIRef* = rdflib.term.URIRef('https://schema.org/mainEntity')

mainEntityOfPage: *URIRef* = rdflib.term.URIRef('https://schema.org/mainEntityOfPage')

maintainer: *URIRef* = rdflib.term.URIRef('https://schema.org/maintainer')

makesOffer: *URIRef* = rdflib.term.URIRef('https://schema.org/makesOffer')

manufacturer: *URIRef* = rdflib.term.URIRef('https://schema.org/manufacturer')

map: *URIRef* = rdflib.term.URIRef('https://schema.org/map')

mapType: *URIRef* = rdflib.term.URIRef('https://schema.org/mapType')

maps: *URIRef* = rdflib.term.URIRef('https://schema.org/maps')

marginOfError: *URIRef* = rdflib.term.URIRef('https://schema.org/marginOfError')

masthead: *URIRef* = rdflib.term.URIRef('https://schema.org/masthead')

material: *URIRef* = rdflib.term.URIRef('https://schema.org/material')

materialExtent: *URIRef* = rdflib.term.URIRef('https://schema.org/materialExtent')

mathExpression: *URIRef* = rdflib.term.URIRef('https://schema.org/mathExpression')

maxPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/maxPrice')

maxValue: *URIRef* = rdflib.term.URIRef('https://schema.org/maxValue')

maximumAttendeeCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/maximumAttendeeCapacity')

maximumEnrollment: *URIRef* = rdflib.term.URIRef('https://schema.org/maximumEnrollment')

maximumIntake: *URIRef* = rdflib.term.URIRef('https://schema.org/maximumIntake')

maximumPhysicalAttendeeCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/maximumPhysicalAttendeeCapacity')

maximumVirtualAttendeeCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/maximumVirtualAttendeeCapacity')

mealService: *URIRef* = rdflib.term.URIRef('https://schema.org/mealService')

measuredProperty: *URIRef* = rdflib.term.URIRef('https://schema.org/measuredProperty')

measuredValue: *URIRef* = rdflib.term.URIRef('https://schema.org/measuredValue')

measurementTechnique: *URIRef* = rdflib.term.URIRef('https://schema.org/measurementTechnique')

mechanismOfAction: *URIRef* = rdflib.term.URIRef('https://schema.org/mechanismOfAction')

mediaAuthenticityCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/mediaAuthenticityCategory')

mediaItemAppearance: *URIRef* = rdflib.term.URIRef('https://schema.org/mediaItemAppearance')

median: *URIRef* = rdflib.term.URIRef('https://schema.org/median')

medicalAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/medicalAudience')

medicalSpecialty: *URIRef* = rdflib.term.URIRef('https://schema.org/medicalSpecialty')

medicineSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/medicineSystem')

meetsEmissionStandard: *URIRef* = rdflib.term.URIRef('https://schema.org/meetsEmissionStandard')

member: *URIRef* = rdflib.term.URIRef('https://schema.org/member')

memberOf: *URIRef* = rdflib.term.URIRef('https://schema.org/memberOf')

members: *URIRef* = rdflib.term.URIRef('https://schema.org/members')

membershipNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/membershipNumber')

membershipPointsEarned: *URIRef* = rdflib.term.URIRef('https://schema.org/membershipPointsEarned')

memoryRequirements: *URIRef* = rdflib.term.URIRef('https://schema.org/memoryRequirements')

mentions: *URIRef* = rdflib.term.URIRef('https://schema.org/mentions')

menu: *URIRef* = rdflib.term.URIRef('https://schema.org/menu')

menuAddOn: *URIRef* = rdflib.term.URIRef('https://schema.org/menuAddOn')

merchant: *URIRef* = rdflib.term.URIRef('https://schema.org/merchant')

merchantReturnDays: *URIRef* = rdflib.term.URIRef('https://schema.org/merchantReturnDays')

merchantReturnLink: *URIRef* = rdflib.term.URIRef('https://schema.org/merchantReturnLink')

messageAttachment: *URIRef* = rdflib.term.URIRef('https://schema.org/messageAttachment')

mileageFromOdometer: *URIRef* = rdflib.term.URIRef('https://schema.org/mileageFromOdometer')

minPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/minPrice')

minValue: *URIRef* = rdflib.term.URIRef('https://schema.org/minValue')

minimumPaymentDue: *URIRef* =
rdflib.term.URIRef('https://schema.org/minimumPaymentDue')

missionCoveragePrioritiesPolicy: *URIRef* =
rdflib.term.URIRef('https://schema.org/missionCoveragePrioritiesPolicy')

model: *URIRef* = rdflib.term.URIRef('https://schema.org/model')

modelDate: *URIRef* = rdflib.term.URIRef('https://schema.org/modelDate')

modifiedTime: *URIRef* = rdflib.term.URIRef('https://schema.org/modifiedTime')

molecularFormula: *URIRef* =
rdflib.term.URIRef('https://schema.org/molecularFormula')

molecularWeight: *URIRef* = rdflib.term.URIRef('https://schema.org/molecularWeight')

monoisotopicMolecularWeight: *URIRef* =
rdflib.term.URIRef('https://schema.org/monoisotopicMolecularWeight')

monthlyMinimumRepaymentAmount: *URIRef* =
rdflib.term.URIRef('https://schema.org/monthlyMinimumRepaymentAmount')

monthsOfExperience: *URIRef* =
rdflib.term.URIRef('https://schema.org/monthsOfExperience')

mpn: *URIRef* = rdflib.term.URIRef('https://schema.org/mpn')

multipleValues: *URIRef* = rdflib.term.URIRef('https://schema.org/multipleValues')

muscleAction: *URIRef* = rdflib.term.URIRef('https://schema.org/muscleAction')

musicArrangement: *URIRef* =
rdflib.term.URIRef('https://schema.org/musicArrangement')

musicBy: *URIRef* = rdflib.term.URIRef('https://schema.org/musicBy')

musicCompositionForm: *URIRef* =
rdflib.term.URIRef('https://schema.org/musicCompositionForm')

musicGroupMember: *URIRef* =
rdflib.term.URIRef('https://schema.org/musicGroupMember')

musicReleaseFormat: *URIRef* =
rdflib.term.URIRef('https://schema.org/musicReleaseFormat')

musicalKey: *URIRef* = rdflib.term.URIRef('https://schema.org/musicalKey')

naics: *URIRef* = rdflib.term.URIRef('https://schema.org/naics')

name: *URIRef* = rdflib.term.URIRef('https://schema.org/name')

namedPosition: *URIRef* = rdflib.term.URIRef('https://schema.org/namedPosition')

nationality: *URIRef* = rdflib.term.URIRef('https://schema.org/nationality')

naturalProgression: *URIRef* = rdflib.term.URIRef('https://schema.org/naturalProgression')

negativeNotes: *URIRef* = rdflib.term.URIRef('https://schema.org/negativeNotes')

nerve: *URIRef* = rdflib.term.URIRef('https://schema.org/nerve')

nerveMotor: *URIRef* = rdflib.term.URIRef('https://schema.org/nerveMotor')

netWorth: *URIRef* = rdflib.term.URIRef('https://schema.org/netWorth')

newsUpdatesAndGuidelines: *URIRef* = rdflib.term.URIRef('https://schema.org/newsUpdatesAndGuidelines')

nextItem: *URIRef* = rdflib.term.URIRef('https://schema.org/nextItem')

noBylinesPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/noBylinesPolicy')

nonEqual: *URIRef* = rdflib.term.URIRef('https://schema.org/nonEqual')

nonProprietaryName: *URIRef* = rdflib.term.URIRef('https://schema.org/nonProprietaryName')

nonprofitStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/nonprofitStatus')

normalRange: *URIRef* = rdflib.term.URIRef('https://schema.org/normalRange')

nsn: *URIRef* = rdflib.term.URIRef('https://schema.org/nsn')

numAdults: *URIRef* = rdflib.term.URIRef('https://schema.org/numAdults')

numChildren: *URIRef* = rdflib.term.URIRef('https://schema.org/numChildren')

numConstraints: *URIRef* = rdflib.term.URIRef('https://schema.org/numConstraints')

numTracks: *URIRef* = rdflib.term.URIRef('https://schema.org/numTracks')

numberOfAccommodationUnits: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfAccommodationUnits')

numberOfAirbags: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfAirbags')

numberOfAvailableAccommodationUnits: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfAvailableAccommodationUnits')

numberOfAxles: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfAxles')

numberOfBathroomsTotal: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfBathroomsTotal')

numberOfBedrooms: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfBedrooms')

numberOfBeds: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfBeds')

numberOfCredits: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfCredits')

numberOfDoors: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfDoors')

numberOfEmployees: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfEmployees')

numberOfEpisodes: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfEpisodes')

numberOfForwardGears: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfForwardGears')

numberOfFullBathrooms: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfFullBathrooms')

numberOfItems: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfItems')

numberOfLoanPayments: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfLoanPayments')

numberOfPages: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfPages')

numberOfPartialBathrooms: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfPartialBathrooms')

numberOfPlayers: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfPlayers')

numberOfPreviousOwners: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfPreviousOwners')

numberOfRooms: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfRooms')

numberOfSeasons: *URIRef* = rdflib.term.URIRef('https://schema.org/numberOfSeasons')

numberedPosition: *URIRef* = rdflib.term.URIRef('https://schema.org/numberedPosition')

nutrition: *URIRef* = rdflib.term.URIRef('https://schema.org/nutrition')

object: *URIRef* = rdflib.term.URIRef('https://schema.org/object')

observationDate: *URIRef* = rdflib.term.URIRef('https://schema.org/observationDate')

observedNode: *URIRef* = rdflib.term.URIRef('https://schema.org/observedNode')

occupancy: *URIRef* = rdflib.term.URIRef('https://schema.org/occupancy')

occupationLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/occupationLocation')

occupationalCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/occupationalCategory')

occupationalCredentialAwarded: *URIRef* = rdflib.term.URIRef('https://schema.org/occupationalCredentialAwarded')

offerCount: *URIRef* = rdflib.term.URIRef('https://schema.org/offerCount')

offeredBy: *URIRef* = rdflib.term.URIRef('https://schema.org/offeredBy')

offers: *URIRef* = rdflib.term.URIRef('https://schema.org/offers')

offersPrescriptionByMail: *URIRef* = rdflib.term.URIRef('https://schema.org/offersPrescriptionByMail')

openingHours: *URIRef* = rdflib.term.URIRef('https://schema.org/openingHours')

openingHoursSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/openingHoursSpecification')

opens: *URIRef* = rdflib.term.URIRef('https://schema.org/opens')

operatingSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/operatingSystem')

opponent: *URIRef* = rdflib.term.URIRef('https://schema.org/opponent')

option: *URIRef* = rdflib.term.URIRef('https://schema.org/option')

orderDate: *URIRef* = rdflib.term.URIRef('https://schema.org/orderDate')

orderDelivery: *URIRef* = rdflib.term.URIRef('https://schema.org/orderDelivery')

orderItemNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/orderItemNumber')

orderItemStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/orderItemStatus')

orderNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/orderNumber')

orderQuantity: *URIRef* = rdflib.term.URIRef('https://schema.org/orderQuantity')

orderStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/orderStatus')

orderedItem: *URIRef* = rdflib.term.URIRef('https://schema.org/orderedItem')

organizer: *URIRef* = rdflib.term.URIRef('https://schema.org/organizer')

originAddress: *URIRef* = rdflib.term.URIRef('https://schema.org/originAddress')

originalMediaContextDescription: *URIRef* = rdflib.term.URIRef('https://schema.org/originalMediaContextDescription')

originalMediaLink: *URIRef* = rdflib.term.URIRef('https://schema.org/originalMediaLink')

originatesFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/originatesFrom')

overdosage: *URIRef* = rdflib.term.URIRef('https://schema.org/overdosage')

ownedFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/ownedFrom')

ownedThrough: *URIRef* = rdflib.term.URIRef('https://schema.org/ownedThrough')

ownershipFundingInfo: *URIRef* = rdflib.term.URIRef('https://schema.org/ownershipFundingInfo')

owns: *URIRef* = rdflib.term.URIRef('https://schema.org/owns')

pageEnd: *URIRef* = rdflib.term.URIRef('https://schema.org/pageEnd')

pageStart: *URIRef* = rdflib.term.URIRef('https://schema.org/pageStart')

pagination: *URIRef* = rdflib.term.URIRef('https://schema.org/pagination')

parent: *URIRef* = rdflib.term.URIRef('https://schema.org/parent')

parentItem: *URIRef* = rdflib.term.URIRef('https://schema.org/parentItem')

parentOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/parentOrganization')

parentService: *URIRef* = rdflib.term.URIRef('https://schema.org/parentService')

parentTaxon: *URIRef* = rdflib.term.URIRef('https://schema.org/parentTaxon')

parents: *URIRef* = rdflib.term.URIRef('https://schema.org/parents')

partOfEpisode: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfEpisode')

partOfInvoice: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfInvoice')

partOfOrder: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfOrder')

partOfSeason: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfSeason')

partOfSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfSeries')

partOfSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfSystem')

partOfTVSeries: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfTVSeries')

partOfTrip: *URIRef* = rdflib.term.URIRef('https://schema.org/partOfTrip')

participant: *URIRef* = rdflib.term.URIRef('https://schema.org/participant')

partySize: *URIRef* = rdflib.term.URIRef('https://schema.org/partySize')

passengerPriorityStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/passengerPriorityStatus')

passengerSequenceNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/passengerSequenceNumber')

pathophysiology: *URIRef* = rdflib.term.URIRef('https://schema.org/pathophysiology')

pattern: *URIRef* = rdflib.term.URIRef('https://schema.org/pattern')

payload: *URIRef* = rdflib.term.URIRef('https://schema.org/payload')

paymentAccepted: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentAccepted')

paymentDue: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentDue')

paymentDueDate: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentDueDate')

paymentMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentMethod')

paymentMethodId: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentMethodId')

paymentStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentStatus')

paymentUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/paymentUrl')

penciler: *URIRef* = rdflib.term.URIRef('https://schema.org/penciler')

percentile10: *URIRef* = rdflib.term.URIRef('https://schema.org/percentile10')

percentile25: *URIRef* = rdflib.term.URIRef('https://schema.org/percentile25')

percentile75: *URIRef* = rdflib.term.URIRef('https://schema.org/percentile75')

percentile90: *URIRef* = rdflib.term.URIRef('https://schema.org/percentile90')

performTime: *URIRef* = rdflib.term.URIRef('https://schema.org/performTime')

performer: *URIRef* = rdflib.term.URIRef('https://schema.org/performer')

performerIn: *URIRef* = rdflib.term.URIRef('https://schema.org/performerIn')

performers: *URIRef* = rdflib.term.URIRef('https://schema.org/performers')

permissionType: *URIRef* = rdflib.term.URIRef('https://schema.org/permissionType')

permissions: *URIRef* = rdflib.term.URIRef('https://schema.org/permissions')

permitAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/permitAudience')

permittedUsage: *URIRef* = rdflib.term.URIRef('https://schema.org/permittedUsage')

petsAllowed: *URIRef* = rdflib.term.URIRef('https://schema.org/petsAllowed')

phoneticText: *URIRef* = rdflib.term.URIRef('https://schema.org/phoneticText')

photo: *URIRef* = rdflib.term.URIRef('https://schema.org/photo')

photos: *URIRef* = rdflib.term.URIRef('https://schema.org/photos')

physicalRequirement: *URIRef* =
rdflib.term.URIRef('https://schema.org/physicalRequirement')

physiologicalBenefits: *URIRef* =
rdflib.term.URIRef('https://schema.org/physiologicalBenefits')

pickupLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/pickupLocation')

pickupTime: *URIRef* = rdflib.term.URIRef('https://schema.org/pickupTime')

playMode: *URIRef* = rdflib.term.URIRef('https://schema.org/playMode')

playerType: *URIRef* = rdflib.term.URIRef('https://schema.org/playerType')

playersOnline: *URIRef* = rdflib.term.URIRef('https://schema.org/playersOnline')

polygon: *URIRef* = rdflib.term.URIRef('https://schema.org/polygon')

populationType: *URIRef* = rdflib.term.URIRef('https://schema.org/populationType')

position: *URIRef* = rdflib.term.URIRef('https://schema.org/position')

positiveNotes: *URIRef* = rdflib.term.URIRef('https://schema.org/positiveNotes')

possibleComplication: *URIRef* =
rdflib.term.URIRef('https://schema.org/possibleComplication')

possibleTreatment: *URIRef* = rdflib.term.URIRef('https://schema.org/possibleTreatment')

postOfficeBoxNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/postOfficeBoxNumber')

postOp: *URIRef* = rdflib.term.URIRef('https://schema.org/postOp')

postalCode: *URIRef* = rdflib.term.URIRef('https://schema.org/postalCode')

postalCodeBegin: *URIRef* = rdflib.term.URIRef('https://schema.org/postalCodeBegin')

postalCodeEnd: *URIRef* = rdflib.term.URIRef('https://schema.org/postalCodeEnd')

postalCodePrefix: *URIRef* = rdflib.term.URIRef('https://schema.org/postalCodePrefix')

postalCodeRange: *URIRef* = rdflib.term.URIRef('https://schema.org/postalCodeRange')

potentialAction: *URIRef* = rdflib.term.URIRef('https://schema.org/potentialAction')

potentialUse: *URIRef* = rdflib.term.URIRef('https://schema.org/potentialUse')

preOp: *URIRef* = rdflib.term.URIRef('https://schema.org/preOp')

predecessorOf: *URIRef* = rdflib.term.URIRef('https://schema.org/predecessorOf')

pregnancyCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/pregnancyCategory')

pregnancyWarning: *URIRef* = rdflib.term.URIRef('https://schema.org/pregnancyWarning')

prepTime: *URIRef* = rdflib.term.URIRef('https://schema.org/prepTime')

preparation: *URIRef* = rdflib.term.URIRef('https://schema.org/preparation')

prescribingInfo: *URIRef* = rdflib.term.URIRef('https://schema.org/prescribingInfo')

prescriptionStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/prescriptionStatus')

previousItem: *URIRef* = rdflib.term.URIRef('https://schema.org/previousItem')

previousStartDate: *URIRef* = rdflib.term.URIRef('https://schema.org/previousStartDate')

price: *URIRef* = rdflib.term.URIRef('https://schema.org/price')

priceComponent: *URIRef* = rdflib.term.URIRef('https://schema.org/priceComponent')

priceComponentType: *URIRef* = rdflib.term.URIRef('https://schema.org/priceComponentType')

priceCurrency: *URIRef* = rdflib.term.URIRef('https://schema.org/priceCurrency')

priceRange: *URIRef* = rdflib.term.URIRef('https://schema.org/priceRange')

priceSpecification: *URIRef* = rdflib.term.URIRef('https://schema.org/priceSpecification')

priceType: *URIRef* = rdflib.term.URIRef('https://schema.org/priceType')

priceValidUntil: *URIRef* = rdflib.term.URIRef('https://schema.org/priceValidUntil')

primaryImageOfPage: *URIRef* = rdflib.term.URIRef('https://schema.org/primaryImageOfPage')

primaryPrevention: *URIRef* = rdflib.term.URIRef('https://schema.org/primaryPrevention')

printColumn: *URIRef* = rdflib.term.URIRef('https://schema.org/printColumn')

printEdition: *URIRef* = rdflib.term.URIRef('https://schema.org/printEdition')

printPage: *URIRef* = rdflib.term.URIRef('https://schema.org/printPage')

printSection: *URIRef* = rdflib.term.URIRef('https://schema.org/printSection')

procedure: *URIRef* = rdflib.term.URIRef('https://schema.org/procedure')

procedureType: *URIRef* = rdflib.term.URIRef('https://schema.org/procedureType')

processingTime: *URIRef* = rdflib.term.URIRef('https://schema.org/processingTime')

processorRequirements: *URIRef* = rdflib.term.URIRef('https://schema.org/processorRequirements')

producer: *URIRef* = rdflib.term.URIRef('https://schema.org/producer')

produces: *URIRef* = rdflib.term.URIRef('https://schema.org/produces')

productGroupID: *URIRef* = rdflib.term.URIRef('https://schema.org/productGroupID')

productID: *URIRef* = rdflib.term.URIRef('https://schema.org/productID')

productSupported: *URIRef* = rdflib.term.URIRef('https://schema.org/productSupported')

productionCompany: *URIRef* = rdflib.term.URIRef('https://schema.org/productionCompany')

productionDate: *URIRef* = rdflib.term.URIRef('https://schema.org/productionDate')

proficiencyLevel: *URIRef* = rdflib.term.URIRef('https://schema.org/proficiencyLevel')

programMembershipUsed: *URIRef* = rdflib.term.URIRef('https://schema.org/programMembershipUsed')

programName: *URIRef* = rdflib.term.URIRef('https://schema.org/programName')

programPrerequisites: *URIRef* = rdflib.term.URIRef('https://schema.org/programPrerequisites')

programType: *URIRef* = rdflib.term.URIRef('https://schema.org/programType')

programmingLanguage: *URIRef* = rdflib.term.URIRef('https://schema.org/programmingLanguage')

programmingModel: *URIRef* = rdflib.term.URIRef('https://schema.org/programmingModel')

propertyID: *URIRef* = rdflib.term.URIRef('https://schema.org/propertyID')

proprietaryName: *URIRef* = rdflib.term.URIRef('https://schema.org/proprietaryName')

proteinContent: *URIRef* = rdflib.term.URIRef('https://schema.org/proteinContent')

provider: *URIRef* = rdflib.term.URIRef('https://schema.org/provider')

providerMobility: *URIRef* = rdflib.term.URIRef('https://schema.org/providerMobility')

providesBroadcastService: *URIRef* = rdflib.term.URIRef('https://schema.org/providesBroadcastService')

providesService: *URIRef* = rdflib.term.URIRef('https://schema.org/providesService')

publicAccess: *URIRef* = rdflib.term.URIRef('https://schema.org/publicAccess')

publicTransportClosuresInfo: *URIRef* = rdflib.term.URIRef('https://schema.org/publicTransportClosuresInfo')

publication: *URIRef* = rdflib.term.URIRef('https://schema.org/publication')

publicationType: *URIRef* = rdflib.term.URIRef('https://schema.org/publicationType')

publishedBy: *URIRef* = rdflib.term.URIRef('https://schema.org/publishedBy')

publishedOn: *URIRef* = rdflib.term.URIRef('https://schema.org/publishedOn')

publisher: *URIRef* = rdflib.term.URIRef('https://schema.org/publisher')

publisherImprint: *URIRef* = rdflib.term.URIRef('https://schema.org/publisherImprint')

publishingPrinciples: *URIRef* = rdflib.term.URIRef('https://schema.org/publishingPrinciples')

purchaseDate: *URIRef* = rdflib.term.URIRef('https://schema.org/purchaseDate')

qualifications: *URIRef* = rdflib.term.URIRef('https://schema.org/qualifications')

quarantineGuidelines: *URIRef* = rdflib.term.URIRef('https://schema.org/quarantineGuidelines')

query: *URIRef* = rdflib.term.URIRef('https://schema.org/query')

quest: *URIRef* = rdflib.term.URIRef('https://schema.org/quest')

question: *URIRef* = rdflib.term.URIRef('https://schema.org/question')

rangeIncludes: *URIRef* = rdflib.term.URIRef('https://schema.org/rangeIncludes')

ratingCount: *URIRef* = rdflib.term.URIRef('https://schema.org/ratingCount')

ratingExplanation: *URIRef* = rdflib.term.URIRef('https://schema.org/ratingExplanation')

ratingValue: *URIRef* = rdflib.term.URIRef('https://schema.org/ratingValue')

readBy: *URIRef* = rdflib.term.URIRef('https://schema.org/readBy')

readonlyValue: *URIRef* = rdflib.term.URIRef('https://schema.org/readonlyValue')

realEstateAgent: *URIRef* = rdflib.term.URIRef('https://schema.org/realEstateAgent')

recipe: *URIRef* = rdflib.term.URIRef('https://schema.org/recipe')

recipeCategory: *URIRef* = rdflib.term.URIRef('https://schema.org/recipeCategory')

recipeCuisine: *URIRef* = rdflib.term.URIRef('https://schema.org/recipeCuisine')

recipeIngredient: *URIRef* = rdflib.term.URIRef('https://schema.org/recipeIngredient')

recipeInstructions: *URIRef* = rdflib.term.URIRef('https://schema.org/recipeInstructions')

recipeYield: *URIRef* = rdflib.term.URIRef('https://schema.org/recipeYield')

recipient: *URIRef* = rdflib.term.URIRef('https://schema.org/recipient')

recognizedBy: *URIRef* = rdflib.term.URIRef('https://schema.org/recognizedBy')

recognizingAuthority: *URIRef* = rdflib.term.URIRef('https://schema.org/recognizingAuthority')

recommendationStrength: *URIRef* = rdflib.term.URIRef('https://schema.org/recommendationStrength')

recommendedIntake: *URIRef* = rdflib.term.URIRef('https://schema.org/recommendedIntake')

recordLabel: *URIRef* = rdflib.term.URIRef('https://schema.org/recordLabel')

recordedAs: *URIRef* = rdflib.term.URIRef('https://schema.org/recordedAs')

recordedAt: *URIRef* = rdflib.term.URIRef('https://schema.org/recordedAt')

recordedIn: *URIRef* = rdflib.term.URIRef('https://schema.org/recordedIn')

recordingOf: *URIRef* = rdflib.term.URIRef('https://schema.org/recordingOf')

recourseLoan: *URIRef* = rdflib.term.URIRef('https://schema.org/recourseLoan')

referenceQuantity: *URIRef* = rdflib.term.URIRef('https://schema.org/referenceQuantity')

referencesOrder: *URIRef* = rdflib.term.URIRef('https://schema.org/referencesOrder')

refundType: *URIRef* = rdflib.term.URIRef('https://schema.org/refundType')

regionDrained: *URIRef* = rdflib.term.URIRef('https://schema.org/regionDrained')

regionsAllowed: *URIRef* = rdflib.term.URIRef('https://schema.org/regionsAllowed')

relatedAnatomy: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedAnatomy')

relatedCondition: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedCondition')

relatedDrug: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedDrug')

relatedLink: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedLink')

relatedStructure: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedStructure')

relatedTherapy: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedTherapy')

relatedTo: *URIRef* = rdflib.term.URIRef('https://schema.org/relatedTo')

releaseDate: *URIRef* = rdflib.term.URIRef('https://schema.org/releaseDate')

releaseNotes: *URIRef* = rdflib.term.URIRef('https://schema.org/releaseNotes')

releaseOf: *URIRef* = rdflib.term.URIRef('https://schema.org/releaseOf')

releasedEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/releasedEvent')

relevantOccupation: *URIRef* = rdflib.term.URIRef('https://schema.org/relevantOccupation')

relevantSpecialty: *URIRef* = rdflib.term.URIRef('https://schema.org/relevantSpecialty')

remainingAttendeeCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/remainingAttendeeCapacity')

renegotiableLoan: *URIRef* = rdflib.term.URIRef('https://schema.org/renegotiableLoan')

repeatCount: *URIRef* = rdflib.term.URIRef('https://schema.org/repeatCount')

repeatFrequency: *URIRef* = rdflib.term.URIRef('https://schema.org/repeatFrequency')

repetitions: *URIRef* = rdflib.term.URIRef('https://schema.org/repetitions')

replacee: *URIRef* = rdflib.term.URIRef('https://schema.org/replacee')

replacer: *URIRef* = rdflib.term.URIRef('https://schema.org/replacer')

replyToUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/replyToUrl')

reportNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/reportNumber')

representativeOfPage: *URIRef* = rdflib.term.URIRef('https://schema.org/representativeOfPage')

requiredCollateral: *URIRef* = rdflib.term.URIRef('https://schema.org/requiredCollateral')

requiredGender: *URIRef* = rdflib.term.URIRef('https://schema.org/requiredGender')

requiredMaxAge: *URIRef* = rdflib.term.URIRef('https://schema.org/requiredMaxAge')

requiredMinAge: *URIRef* = rdflib.term.URIRef('https://schema.org/requiredMinAge')

requiredQuantity: *URIRef* =
rdflib.term.URIRef('https://schema.org/requiredQuantity')

requirements: *URIRef* = rdflib.term.URIRef('https://schema.org/requirements')

requiresSubscription: *URIRef* =
rdflib.term.URIRef('https://schema.org/requiresSubscription')

reservationFor: *URIRef* = rdflib.term.URIRef('https://schema.org/reservationFor')

reservationId: *URIRef* = rdflib.term.URIRef('https://schema.org/reservationId')

reservationStatus: *URIRef* =
rdflib.term.URIRef('https://schema.org/reservationStatus')

reservedTicket: *URIRef* = rdflib.term.URIRef('https://schema.org/reservedTicket')

responsibilities: *URIRef* =
rdflib.term.URIRef('https://schema.org/responsibilities')

restPeriods: *URIRef* = rdflib.term.URIRef('https://schema.org/restPeriods')

restockingFee: *URIRef* = rdflib.term.URIRef('https://schema.org/restockingFee')

result: *URIRef* = rdflib.term.URIRef('https://schema.org/result')

resultComment: *URIRef* = rdflib.term.URIRef('https://schema.org/resultComment')

resultReview: *URIRef* = rdflib.term.URIRef('https://schema.org/resultReview')

returnFees: *URIRef* = rdflib.term.URIRef('https://schema.org/returnFees')

returnLabelSource: *URIRef* =
rdflib.term.URIRef('https://schema.org/returnLabelSource')

returnMethod: *URIRef* = rdflib.term.URIRef('https://schema.org/returnMethod')

returnPolicyCategory: *URIRef* =
rdflib.term.URIRef('https://schema.org/returnPolicyCategory')

returnPolicyCountry: *URIRef* =
rdflib.term.URIRef('https://schema.org/returnPolicyCountry')

returnPolicySeasonalOverride: *URIRef* =
rdflib.term.URIRef('https://schema.org/returnPolicySeasonalOverride')

returnShippingFeesAmount: *URIRef* =
rdflib.term.URIRef('https://schema.org/returnShippingFeesAmount')

review: *URIRef* = rdflib.term.URIRef('https://schema.org/review')

reviewAspect: *URIRef* = rdflib.term.URIRef('https://schema.org/reviewAspect')

reviewBody: *URIRef* = rdflib.term.URIRef('https://schema.org/reviewBody')

reviewCount: *URIRef* = rdflib.term.URIRef('https://schema.org/reviewCount')

reviewRating: *URIRef* = rdflib.term.URIRef('https://schema.org/reviewRating')

reviewedBy: *URIRef* = rdflib.term.URIRef('https://schema.org/reviewedBy')

reviews: *URIRef* = rdflib.term.URIRef('https://schema.org/reviews')

riskFactor: *URIRef* = rdflib.term.URIRef('https://schema.org/riskFactor')

risks: *URIRef* = rdflib.term.URIRef('https://schema.org/risks')

roleName: *URIRef* = rdflib.term.URIRef('https://schema.org/roleName')

roofLoad: *URIRef* = rdflib.term.URIRef('https://schema.org/roofLoad')

rsvpResponse: *URIRef* = rdflib.term.URIRef('https://schema.org/rsvpResponse')

runsTo: *URIRef* = rdflib.term.URIRef('https://schema.org/runsTo')

runtime: *URIRef* = rdflib.term.URIRef('https://schema.org/runtime')

runtimePlatform: *URIRef* = rdflib.term.URIRef('https://schema.org/runtimePlatform')

rxcui: *URIRef* = rdflib.term.URIRef('https://schema.org/rxcui')

safetyConsideration: *URIRef* =
rdflib.term.URIRef('https://schema.org/safetyConsideration')

salaryCurrency: *URIRef* = rdflib.term.URIRef('https://schema.org/salaryCurrency')

salaryUponCompletion: *URIRef* =
rdflib.term.URIRef('https://schema.org/salaryUponCompletion')

sameAs: *URIRef* = rdflib.term.URIRef('https://schema.org/sameAs')

sampleType: *URIRef* = rdflib.term.URIRef('https://schema.org/sampleType')

saturatedFatContent: *URIRef* =
rdflib.term.URIRef('https://schema.org/saturatedFatContent')

scheduleTimezone: *URIRef* =
rdflib.term.URIRef('https://schema.org/scheduleTimezone')

scheduledPaymentDate: *URIRef* =
rdflib.term.URIRef('https://schema.org/scheduledPaymentDate')

scheduledTime: *URIRef* = rdflib.term.URIRef('https://schema.org/scheduledTime')

schemaVersion: *URIRef* = rdflib.term.URIRef('https://schema.org/schemaVersion')

schoolClosuresInfo: *URIRef* =
rdflib.term.URIRef('https://schema.org/schoolClosuresInfo')

screenCount: *URIRef* = rdflib.term.URIRef('https://schema.org/screenCount')

screenshot: *URIRef* = rdflib.term.URIRef('https://schema.org/screenshot')

sdDatePublished: *URIRef* = rdflib.term.URIRef('https://schema.org/sdDatePublished')

sdLicense: *URIRef* = rdflib.term.URIRef('https://schema.org/sdLicense')

sdPublisher: *URIRef* = rdflib.term.URIRef('https://schema.org/sdPublisher')

season: *URIRef* = rdflib.term.URIRef('https://schema.org/season')

seasonNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/seasonNumber')

seasons: *URIRef* = rdflib.term.URIRef('https://schema.org/seasons')

seatNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/seatNumber')

seatRow: *URIRef* = rdflib.term.URIRef('https://schema.org/seatRow')

seatSection: *URIRef* = rdflib.term.URIRef('https://schema.org/seatSection')

seatingCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/seatingCapacity')

seatingType: *URIRef* = rdflib.term.URIRef('https://schema.org/seatingType')

secondaryPrevention: *URIRef* =
rdflib.term.URIRef('https://schema.org/secondaryPrevention')

securityClearanceRequirement: *URIRef* =
rdflib.term.URIRef('https://schema.org/securityClearanceRequirement')

securityScreening: *URIRef* =
rdflib.term.URIRef('https://schema.org/securityScreening')

seeks: *URIRef* = rdflib.term.URIRef('https://schema.org/seeks')

seller: *URIRef* = rdflib.term.URIRef('https://schema.org/seller')

sender: *URIRef* = rdflib.term.URIRef('https://schema.org/sender')

sensoryRequirement: *URIRef* =
rdflib.term.URIRef('https://schema.org/sensoryRequirement')

sensoryUnit: *URIRef* = rdflib.term.URIRef('https://schema.org/sensoryUnit')

serialNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/serialNumber')

seriousAdverseOutcome: *URIRef* =
rdflib.term.URIRef('https://schema.org/seriousAdverseOutcome')

serverStatus: *URIRef* = rdflib.term.URIRef('https://schema.org/serverStatus')

servesCuisine: *URIRef* = rdflib.term.URIRef('https://schema.org/servesCuisine')

serviceArea: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceArea')

serviceAudience: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceAudience')

serviceLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceLocation')

serviceOperator: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceOperator')

serviceOutput: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceOutput')

servicePhone: *URIRef* = rdflib.term.URIRef('https://schema.org/servicePhone')

servicePostalAddress: *URIRef* = rdflib.term.URIRef('https://schema.org/servicePostalAddress')

serviceSmsNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceSmsNumber')

serviceType: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceType')

serviceUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/serviceUrl')

servingSize: *URIRef* = rdflib.term.URIRef('https://schema.org/servingSize')

sha256: *URIRef* = rdflib.term.URIRef('https://schema.org/sha256')

sharedContent: *URIRef* = rdflib.term.URIRef('https://schema.org/sharedContent')

shippingDestination: *URIRef* = rdflib.term.URIRef('https://schema.org/shippingDestination')

shippingDetails: *URIRef* = rdflib.term.URIRef('https://schema.org/shippingDetails')

shippingLabel: *URIRef* = rdflib.term.URIRef('https://schema.org/shippingLabel')

shippingRate: *URIRef* = rdflib.term.URIRef('https://schema.org/shippingRate')

shippingSettingsLink: *URIRef* = rdflib.term.URIRef('https://schema.org/shippingSettingsLink')

sibling: *URIRef* = rdflib.term.URIRef('https://schema.org/sibling')

siblings: *URIRef* = rdflib.term.URIRef('https://schema.org/siblings')

signDetected: *URIRef* = rdflib.term.URIRef('https://schema.org/signDetected')

signOrSymptom: *URIRef* = rdflib.term.URIRef('https://schema.org/signOrSymptom')

significance: *URIRef* = rdflib.term.URIRef('https://schema.org/significance')

significantLink: *URIRef* = rdflib.term.URIRef('https://schema.org/significantLink')

significantLinks: *URIRef* = rdflib.term.URIRef('https://schema.org/significantLinks')

size: *URIRef* = rdflib.term.URIRef('https://schema.org/size')

sizeGroup: *URIRef* = rdflib.term.URIRef('https://schema.org/sizeGroup')

sizeSystem: *URIRef* = rdflib.term.URIRef('https://schema.org/sizeSystem')

skills: *URIRef* = rdflib.term.URIRef('https://schema.org/skills')

sku: *URIRef* = rdflib.term.URIRef('https://schema.org/sku')

slogan: *URIRef* = rdflib.term.URIRef('https://schema.org/slogan')

smiles: *URIRef* = rdflib.term.URIRef('https://schema.org/smiles')

smokingAllowed: *URIRef* = rdflib.term.URIRef('https://schema.org/smokingAllowed')

sodiumContent: *URIRef* = rdflib.term.URIRef('https://schema.org/sodiumContent')

softwareAddOn: *URIRef* = rdflib.term.URIRef('https://schema.org/softwareAddOn')

softwareHelp: *URIRef* = rdflib.term.URIRef('https://schema.org/softwareHelp')

softwareRequirements: *URIRef* =
rdflib.term.URIRef('https://schema.org/softwareRequirements')

softwareVersion: *URIRef* = rdflib.term.URIRef('https://schema.org/softwareVersion')

sourceOrganization: *URIRef* =
rdflib.term.URIRef('https://schema.org/sourceOrganization')

sourcedFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/sourcedFrom')

spatial: *URIRef* = rdflib.term.URIRef('https://schema.org/spatial')

spatialCoverage: *URIRef* = rdflib.term.URIRef('https://schema.org/spatialCoverage')

speakable: *URIRef* = rdflib.term.URIRef('https://schema.org/speakable')

specialCommitments: *URIRef* =
rdflib.term.URIRef('https://schema.org/specialCommitments')

specialOpeningHoursSpecification: *URIRef* =
rdflib.term.URIRef('https://schema.org/specialOpeningHoursSpecification')

specialty: *URIRef* = rdflib.term.URIRef('https://schema.org/specialty')

speechToTextMarkup: *URIRef* =
rdflib.term.URIRef('https://schema.org/speechToTextMarkup')

speed: *URIRef* = rdflib.term.URIRef('https://schema.org/speed')

spokenByCharacter: *URIRef* =
rdflib.term.URIRef('https://schema.org/spokenByCharacter')

sponsor: *URIRef* = rdflib.term.URIRef('https://schema.org/sponsor')

sport: *URIRef* = rdflib.term.URIRef('https://schema.org/sport')

sportsActivityLocation: *URIRef* =
rdflib.term.URIRef('https://schema.org/sportsActivityLocation')

sportsEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/sportsEvent')

sportsTeam: *URIRef* = rdflib.term.URIRef('https://schema.org/sportsTeam')

spouse: *URIRef* = rdflib.term.URIRef('https://schema.org/spouse')

stage: *URIRef* = rdflib.term.URIRef('https://schema.org/stage')

stageAsNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/stageAsNumber')

starRating: *URIRef* = rdflib.term.URIRef('https://schema.org/starRating')

startDate: *URIRef* = rdflib.term.URIRef('https://schema.org/startDate')

startOffset: *URIRef* = rdflib.term.URIRef('https://schema.org/startOffset')

startTime: *URIRef* = rdflib.term.URIRef('https://schema.org/startTime')

status: *URIRef* = rdflib.term.URIRef('https://schema.org/status')

steeringPosition: *URIRef* =
rdflib.term.URIRef('https://schema.org/steeringPosition')

step: *URIRef* = rdflib.term.URIRef('https://schema.org/step')

stepValue: *URIRef* = rdflib.term.URIRef('https://schema.org/stepValue')

steps: *URIRef* = rdflib.term.URIRef('https://schema.org/steps')

storageRequirements: *URIRef* =
rdflib.term.URIRef('https://schema.org/storageRequirements')

streetAddress: *URIRef* = rdflib.term.URIRef('https://schema.org/streetAddress')

strengthUnit: *URIRef* = rdflib.term.URIRef('https://schema.org/strengthUnit')

strengthValue: *URIRef* = rdflib.term.URIRef('https://schema.org/strengthValue')

structuralClass: *URIRef* = rdflib.term.URIRef('https://schema.org/structuralClass')

study: *URIRef* = rdflib.term.URIRef('https://schema.org/study')

studyDesign: *URIRef* = rdflib.term.URIRef('https://schema.org/studyDesign')

studyLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/studyLocation')

studySubject: *URIRef* = rdflib.term.URIRef('https://schema.org/studySubject')

subEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/subEvent')

subEvents: *URIRef* = rdflib.term.URIRef('https://schema.org/subEvents')

subOrganization: *URIRef* = rdflib.term.URIRef('https://schema.org/subOrganization')

subReservation: *URIRef* = rdflib.term.URIRef('https://schema.org/subReservation')

subStageSuffix: *URIRef* = rdflib.term.URIRef('https://schema.org/subStageSuffix')

subStructure: *URIRef* = rdflib.term.URIRef('https://schema.org/subStructure')

subTest: *URIRef* = rdflib.term.URIRef('https://schema.org/subTest')

subTrip: *URIRef* = rdflib.term.URIRef('https://schema.org/subTrip')

subjectOf: *URIRef* = rdflib.term.URIRef('https://schema.org/subjectOf')

subtitleLanguage: *URIRef* =
rdflib.term.URIRef('https://schema.org/subtitleLanguage')

successorOf: *URIRef* = rdflib.term.URIRef('https://schema.org/successorOf')

sugarContent: *URIRef* = rdflib.term.URIRef('https://schema.org/sugarContent')

suggestedAge: *URIRef* = rdflib.term.URIRef('https://schema.org/suggestedAge')

suggestedAnswer: *URIRef* = rdflib.term.URIRef('https://schema.org/suggestedAnswer')

suggestedGender: *URIRef* = rdflib.term.URIRef('https://schema.org/suggestedGender')

suggestedMaxAge: *URIRef* = rdflib.term.URIRef('https://schema.org/suggestedMaxAge')

suggestedMeasurement: *URIRef* =
rdflib.term.URIRef('https://schema.org/suggestedMeasurement')

suggestedMinAge: *URIRef* = rdflib.term.URIRef('https://schema.org/suggestedMinAge')

suitableForDiet: *URIRef* = rdflib.term.URIRef('https://schema.org/suitableForDiet')

superEvent: *URIRef* = rdflib.term.URIRef('https://schema.org/superEvent')

supersededBy: *URIRef* = rdflib.term.URIRef('https://schema.org/supersededBy')

supply: *URIRef* = rdflib.term.URIRef('https://schema.org/supply')

supplyTo: *URIRef* = rdflib.term.URIRef('https://schema.org/supplyTo')

supportingData: *URIRef* = rdflib.term.URIRef('https://schema.org/supportingData')

surface: *URIRef* = rdflib.term.URIRef('https://schema.org/surface')

target: *URIRef* = rdflib.term.URIRef('https://schema.org/target')

targetCollection: *URIRef* =
rdflib.term.URIRef('https://schema.org/targetCollection')

targetDescription: *URIRef* =
rdflib.term.URIRef('https://schema.org/targetDescription')

targetName: *URIRef* = rdflib.term.URIRef('https://schema.org/targetName')

targetPlatform: *URIRef* = rdflib.term.URIRef('https://schema.org/targetPlatform')

targetPopulation: *URIRef* =
rdflib.term.URIRef('https://schema.org/targetPopulation')

targetProduct: *URIRef* = rdflib.term.URIRef('https://schema.org/targetProduct')

targetUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/targetUrl')

taxID: *URIRef* = rdflib.term.URIRef('https://schema.org/taxID')

taxonRank: *URIRef* = rdflib.term.URIRef('https://schema.org/taxonRank')

taxonomicRange: *URIRef* = rdflib.term.URIRef('https://schema.org/taxonomicRange')

teaches: *URIRef* = rdflib.term.URIRef('https://schema.org/teaches')

telephone: *URIRef* = rdflib.term.URIRef('https://schema.org/telephone')

temporal: *URIRef* = rdflib.term.URIRef('https://schema.org/temporal')

temporalCoverage: *URIRef* =
rdflib.term.URIRef('https://schema.org/temporalCoverage')

termCode: *URIRef* = rdflib.term.URIRef('https://schema.org/termCode')

termDuration: *URIRef* = rdflib.term.URIRef('https://schema.org/termDuration')

termsOfService: *URIRef* = rdflib.term.URIRef('https://schema.org/termsOfService')

termsPerYear: *URIRef* = rdflib.term.URIRef('https://schema.org/termsPerYear')

text: *URIRef* = rdflib.term.URIRef('https://schema.org/text')

textValue: *URIRef* = rdflib.term.URIRef('https://schema.org/textValue')

thumbnail: *URIRef* = rdflib.term.URIRef('https://schema.org/thumbnail')

thumbnailUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/thumbnailUrl')

tickerSymbol: *URIRef* = rdflib.term.URIRef('https://schema.org/tickerSymbol')

ticketNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/ticketNumber')

ticketToken: *URIRef* = rdflib.term.URIRef('https://schema.org/ticketToken')

ticketedSeat: *URIRef* = rdflib.term.URIRef('https://schema.org/ticketedSeat')

timeOfDay: *URIRef* = rdflib.term.URIRef('https://schema.org/timeOfDay')

timeRequired: *URIRef* = rdflib.term.URIRef('https://schema.org/timeRequired')

timeToComplete: *URIRef* = rdflib.term.URIRef('https://schema.org/timeToComplete')

tissueSample: *URIRef* = rdflib.term.URIRef('https://schema.org/tissueSample')

title: *URIRef* = rdflib.term.URIRef('https://schema.org/title')

titleEIDR: *URIRef* = rdflib.term.URIRef('https://schema.org/titleEIDR')

toLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/toLocation')

toRecipient: *URIRef* = rdflib.term.URIRef('https://schema.org/toRecipient')

tocContinuation: *URIRef* = rdflib.term.URIRef('https://schema.org/tocContinuation')

tocEntry: *URIRef* = rdflib.term.URIRef('https://schema.org/tocEntry')

tongueWeight: *URIRef* = rdflib.term.URIRef('https://schema.org/tongueWeight')

tool: *URIRef* = rdflib.term.URIRef('https://schema.org/tool')

torque: *URIRef* = rdflib.term.URIRef('https://schema.org/torque')

totalJobOpenings: *URIRef* =
rdflib.term.URIRef('https://schema.org/totalJobOpenings')

totalPaymentDue: *URIRef* = rdflib.term.URIRef('https://schema.org/totalPaymentDue')

totalPrice: *URIRef* = rdflib.term.URIRef('https://schema.org/totalPrice')

totalTime: *URIRef* = rdflib.term.URIRef('https://schema.org/totalTime')

tourBookingPage: *URIRef* = rdflib.term.URIRef('https://schema.org/tourBookingPage')

touristType: *URIRef* = rdflib.term.URIRef('https://schema.org/touristType')

track: *URIRef* = rdflib.term.URIRef('https://schema.org/track')

trackingNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/trackingNumber')

trackingUrl: *URIRef* = rdflib.term.URIRef('https://schema.org/trackingUrl')

tracks: *URIRef* = rdflib.term.URIRef('https://schema.org/tracks')

trailer: *URIRef* = rdflib.term.URIRef('https://schema.org/trailer')

trailerWeight: *URIRef* = rdflib.term.URIRef('https://schema.org/trailerWeight')

trainName: *URIRef* = rdflib.term.URIRef('https://schema.org/trainName')

trainNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/trainNumber')

trainingSalary: *URIRef* = rdflib.term.URIRef('https://schema.org/trainingSalary')

transFatContent: *URIRef* = rdflib.term.URIRef('https://schema.org/transFatContent')

transcript: *URIRef* = rdflib.term.URIRef('https://schema.org/transcript')

transitTime: *URIRef* = rdflib.term.URIRef('https://schema.org/transitTime')

transitTimeLabel: *URIRef* =
rdflib.term.URIRef('https://schema.org/transitTimeLabel')

translationOfWork: *URIRef* =
rdflib.term.URIRef('https://schema.org/translationOfWork')

translator: *URIRef* = rdflib.term.URIRef('https://schema.org/translator')

transmissionMethod: *URIRef* =
rdflib.term.URIRef('https://schema.org/transmissionMethod')

travelBans: *URIRef* = rdflib.term.URIRef('https://schema.org/travelBans')

trialDesign: *URIRef* = rdflib.term.URIRef('https://schema.org/trialDesign')

tributary: *URIRef* = rdflib.term.URIRef('https://schema.org/tributary')

typeOfBed: *URIRef* = rdflib.term.URIRef('https://schema.org/typeOfBed')

typeOfGood: *URIRef* = rdflib.term.URIRef('https://schema.org/typeOfGood')

typicalAgeRange: *URIRef* = rdflib.term.URIRef('https://schema.org/typicalAgeRange')

typicalCreditsPerTerm: *URIRef* =
rdflib.term.URIRef('https://schema.org/typicalCreditsPerTerm')

typicalTest: *URIRef* = rdflib.term.URIRef('https://schema.org/typicalTest')

underName: *URIRef* = rdflib.term.URIRef('https://schema.org/underName')

unitCode: *URIRef* = rdflib.term.URIRef('https://schema.org/unitCode')

unitText: *URIRef* = rdflib.term.URIRef('https://schema.org/unitText')

unnamedSourcesPolicy: *URIRef* =
rdflib.term.URIRef('https://schema.org/unnamedSourcesPolicy')

unsaturatedFatContent: *URIRef* = rdflib.term.URIRef('https://schema.org/unsaturatedFatContent')

uploadDate: *URIRef* = rdflib.term.URIRef('https://schema.org/uploadDate')

upvoteCount: *URIRef* = rdflib.term.URIRef('https://schema.org/upvoteCount')

url: *URIRef* = rdflib.term.URIRef('https://schema.org/url')

urlTemplate: *URIRef* = rdflib.term.URIRef('https://schema.org/urlTemplate')

usageInfo: *URIRef* = rdflib.term.URIRef('https://schema.org/usageInfo')

usedToDiagnose: *URIRef* = rdflib.term.URIRef('https://schema.org/usedToDiagnose')

userInteractionCount: *URIRef* = rdflib.term.URIRef('https://schema.org/userInteractionCount')

usesDevice: *URIRef* = rdflib.term.URIRef('https://schema.org/usesDevice')

usesHealthPlanIdStandard: *URIRef* = rdflib.term.URIRef('https://schema.org/usesHealthPlanIdStandard')

utterances: *URIRef* = rdflib.term.URIRef('https://schema.org/utterances')

validFor: *URIRef* = rdflib.term.URIRef('https://schema.org/validFor')

validFrom: *URIRef* = rdflib.term.URIRef('https://schema.org/validFrom')

validIn: *URIRef* = rdflib.term.URIRef('https://schema.org/validIn')

validThrough: *URIRef* = rdflib.term.URIRef('https://schema.org/validThrough')

validUntil: *URIRef* = rdflib.term.URIRef('https://schema.org/validUntil')

value: *URIRef* = rdflib.term.URIRef('https://schema.org/value')

valueAddedTaxIncluded: *URIRef* = rdflib.term.URIRef('https://schema.org/valueAddedTaxIncluded')

valueMaxLength: *URIRef* = rdflib.term.URIRef('https://schema.org/valueMaxLength')

valueMinLength: *URIRef* = rdflib.term.URIRef('https://schema.org/valueMinLength')

valueName: *URIRef* = rdflib.term.URIRef('https://schema.org/valueName')

valuePattern: *URIRef* = rdflib.term.URIRef('https://schema.org/valuePattern')

valueReference: *URIRef* = rdflib.term.URIRef('https://schema.org/valueReference')

valueRequired: *URIRef* = rdflib.term.URIRef('https://schema.org/valueRequired')

variableMeasured: *URIRef* = rdflib.term.URIRef('https://schema.org/variableMeasured')

variantCover: *URIRef* = rdflib.term.URIRef('https://schema.org/variantCover')

variesBy: *URIRef* = rdflib.term.URIRef('https://schema.org/variesBy')

vatID: *URIRef* = rdflib.term.URIRef('https://schema.org/vatID')

vehicleConfiguration: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleConfiguration')

vehicleEngine: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleEngine')

vehicleIdentificationNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleIdentificationNumber')

vehicleInteriorColor: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleInteriorColor')

vehicleInteriorType: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleInteriorType')

vehicleModelDate: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleModelDate')

vehicleSeatingCapacity: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleSeatingCapacity')

vehicleSpecialUsage: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleSpecialUsage')

vehicleTransmission: *URIRef* = rdflib.term.URIRef('https://schema.org/vehicleTransmission')

vendor: *URIRef* = rdflib.term.URIRef('https://schema.org/vendor')

verificationFactCheckingPolicy: *URIRef* = rdflib.term.URIRef('https://schema.org/verificationFactCheckingPolicy')

version: *URIRef* = rdflib.term.URIRef('https://schema.org/version')

video: *URIRef* = rdflib.term.URIRef('https://schema.org/video')

videoFormat: *URIRef* = rdflib.term.URIRef('https://schema.org/videoFormat')

videoFrameSize: *URIRef* = rdflib.term.URIRef('https://schema.org/videoFrameSize')

videoQuality: *URIRef* = rdflib.term.URIRef('https://schema.org/videoQuality')

volumeNumber: *URIRef* = rdflib.term.URIRef('https://schema.org/volumeNumber')

warning: *URIRef* = rdflib.term.URIRef('https://schema.org/warning')

warranty: *URIRef* = rdflib.term.URIRef('https://schema.org/warranty')

warrantyPromise: *URIRef* = rdflib.term.URIRef('https://schema.org/warrantyPromise')

warrantyScope: *URIRef* = rdflib.term.URIRef('https://schema.org/warrantyScope')

webCheckinTime: *URIRef* = rdflib.term.URIRef('https://schema.org/webCheckinTime')

webFeed: *URIRef* = rdflib.term.URIRef('https://schema.org/webFeed')

weight: *URIRef* = rdflib.term.URIRef('https://schema.org/weight')

weightTotal: *URIRef* = rdflib.term.URIRef('https://schema.org/weightTotal')

wheelbase: *URIRef* = rdflib.term.URIRef('https://schema.org/wheelbase')

width: *URIRef* = rdflib.term.URIRef('https://schema.org/width')

winner: *URIRef* = rdflib.term.URIRef('https://schema.org/winner')

wordCount: *URIRef* = rdflib.term.URIRef('https://schema.org/wordCount')

workExample: *URIRef* = rdflib.term.URIRef('https://schema.org/workExample')

workFeatured: *URIRef* = rdflib.term.URIRef('https://schema.org/workFeatured')

workHours: *URIRef* = rdflib.term.URIRef('https://schema.org/workHours')

workLocation: *URIRef* = rdflib.term.URIRef('https://schema.org/workLocation')

workPerformed: *URIRef* = rdflib.term.URIRef('https://schema.org/workPerformed')

workPresented: *URIRef* = rdflib.term.URIRef('https://schema.org/workPresented')

workTranslation: *URIRef* = rdflib.term.URIRef('https://schema.org/workTranslation')

workload: *URIRef* = rdflib.term.URIRef('https://schema.org/workload')

worksFor: *URIRef* = rdflib.term.URIRef('https://schema.org/worksFor')

worstRating: *URIRef* = rdflib.term.URIRef('https://schema.org/worstRating')

xpath: *URIRef* = rdflib.term.URIRef('https://schema.org/xpath')

yearBuilt: *URIRef* = rdflib.term.URIRef('https://schema.org/yearBuilt')

yearlyRevenue: *URIRef* = rdflib.term.URIRef('https://schema.org/yearlyRevenue')

yearsInOperation: *URIRef* = rdflib.term.URIRef('https://schema.org/yearsInOperation')

class rdflib.SH

Bases: *DefinedNamespace*

W3C Shapes Constraint Language (SHACL) Vocabulary

This vocabulary defines terms used in SHACL, the W3C Shapes Constraint Language.

Generated from: https://www.w3.org/ns/shacl.ttl Date: 2020-05-26 14:20:08.041103

AbstractResult: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#AbstractResult')

AndConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#AndConstraintComponent')

BlankNode: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#BlankNode')

BlankNodeOrIRI: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#BlankNodeOrIRI')

BlankNodeOrLiteral: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#BlankNodeOrLiteral')

ClassConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ClassConstraintComponent')

ClosedConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ClosedConstraintComponent')

ConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ConstraintComponent')

DatatypeConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#DatatypeConstraintComponent')

DisjointConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#DisjointConstraintComponent')

EqualsConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#EqualsConstraintComponent')

ExpressionConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ExpressionConstraintComponent')

Function: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Function')

HasValueConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#HasValueConstraintComponent')

IRI: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#IRI')

IRIOrLiteral: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#IRIOrLiteral')

InConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#InConstraintComponent')

Info: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Info')

JSConstraint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSConstraint')

JSConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSConstraintComponent')

JSExecutable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSExecutable')

JSFunction: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSFunction')

JSLibrary: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSLibrary')

JSRule: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSRule')

JSTarget: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSTarget')

JSTargetType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSTargetType')

JSValidator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#JSValidator')

LanguageInConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#LanguageInConstraintComponent')

LessThanConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#LessThanConstraintComponent')

LessThanOrEqualsConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#LessThanOrEqualsConstraintComponent')

Literal: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Literal')

MaxCountConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MaxCountConstraintComponent')

MaxExclusiveConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MaxExclusiveConstraintComponent')

MaxInclusiveConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MaxInclusiveConstraintComponent')

MaxLengthConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MaxLengthConstraintComponent')

MinCountConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MinCountConstraintComponent')

MinExclusiveConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MinExclusiveConstraintComponent')

MinInclusiveConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MinInclusiveConstraintComponent')

MinLengthConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#MinLengthConstraintComponent')

NodeConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#NodeConstraintComponent')

NodeKind: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#NodeKind')

NodeKindConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#NodeKindConstraintComponent')

NodeShape: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#NodeShape')

NotConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#NotConstraintComponent')

OrConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#OrConstraintComponent')

Parameter: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Parameter')

Parameterizable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Parameterizable')

PatternConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#PatternConstraintComponent')

PrefixDeclaration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#PrefixDeclaration')

PropertyConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#PropertyConstraintComponent')

PropertyGroup: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#PropertyGroup')

PropertyShape: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#PropertyShape')

QualifiedMaxCountConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#QualifiedMaxCountConstraintComponent')

QualifiedMinCountConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#QualifiedMinCountConstraintComponent')

ResultAnnotation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ResultAnnotation')

Rule: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Rule')

SPARQLAskExecutable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLAskExecutable')

SPARQLAskValidator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLAskValidator')

SPARQLConstraint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLConstraint')

SPARQLConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLConstraintComponent')

SPARQLConstructExecutable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLConstructExecutable')

SPARQLExecutable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLExecutable')

SPARQLFunction: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLFunction')

SPARQLRule: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLRule')

SPARQLSelectExecutable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLSelectExecutable')

SPARQLSelectValidator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLSelectValidator')

SPARQLTarget: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLTarget')

SPARQLTargetType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLTargetType')

SPARQLUpdateExecutable: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#SPARQLUpdateExecutable')

Severity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Severity')

Shape: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Shape')

Target: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Target')

TargetType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#TargetType')

TripleRule: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#TripleRule')

UniqueLangConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#UniqueLangConstraintComponent')

ValidationReport: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ValidationReport')

ValidationResult: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ValidationResult')

Validator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Validator')

Violation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Violation')

Warning: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#Warning')

XoneConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#XoneConstraintComponent')

alternativePath: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#alternativePath')

annotationProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#annotationProperty')

annotationValue: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#annotationValue')

annotationVarName: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#annotationVarName')

ask: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ask')

closed: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#closed')

condition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#condition')

conforms: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#conforms')

construct: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#construct')

datatype: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#datatype')

deactivated: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#deactivated')

declare: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#declare')

defaultValue: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#defaultValue')

description: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#description')

detail: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#detail')

disjoint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#disjoint')

entailment: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#entailment')

equals: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#equals')

expression: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#expression')

filterShape: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#filterShape')

flags: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#flags')

focusNode: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#focusNode')

group: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#group')

hasValue: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#hasValue')

ignoredProperties: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#ignoredProperties')

intersection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#intersection')

inversePath: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#inversePath')

js: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#js')

jsFunctionName: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#jsFunctionName')

jsLibrary: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#jsLibrary')

jsLibraryURL: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#jsLibraryURL')

labelTemplate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#labelTemplate')

languageIn: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#languageIn')

lessThan: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#lessThan')

lessThanOrEquals: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#lessThanOrEquals')

maxCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#maxCount')

maxExclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#maxExclusive')

maxInclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#maxInclusive')

maxLength: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#maxLength')

message: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#message')

minCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#minCount')

minExclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#minExclusive')

minInclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#minInclusive')

minLength: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#minLength')

name: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#name')

namespace: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#namespace')

node: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#node')

nodeKind: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#nodeKind')

nodeValidator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#nodeValidator')

nodes: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#nodes')

object: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#object')

oneOrMorePath: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#oneOrMorePath')

optional: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#optional')

order: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#order')

parameter: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#parameter')

path: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#path')

pattern: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#pattern')

predicate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#predicate')

prefix: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#prefix')

prefixes: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#prefixes')

property: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#property')

propertyValidator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#propertyValidator')

qualifiedMaxCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#qualifiedMaxCount')

qualifiedMinCount: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#qualifiedMinCount')

qualifiedValueShape: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#qualifiedValueShape')

qualifiedValueShapesDisjoint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#qualifiedValueShapesDisjoint')

result: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#result')

resultAnnotation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#resultAnnotation')

resultMessage: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#resultMessage')

resultPath: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#resultPath')

resultSeverity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#resultSeverity')

returnType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#returnType')

rule: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#rule')

select: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#select')

severity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#severity')

shapesGraph: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#shapesGraph')

shapesGraphWellFormed: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#shapesGraphWellFormed')

sourceConstraint: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#sourceConstraint')

sourceConstraintComponent: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#sourceConstraintComponent')

sourceShape: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#sourceShape')

sparql: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#sparql')

subject: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#subject')

suggestedShapesGraph: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#suggestedShapesGraph')

target: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#target')

targetClass: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#targetClass')

targetNode: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#targetNode')

targetObjectsOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#targetObjectsOf')

targetSubjectsOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#targetSubjectsOf')

this: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#this')

union: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#union')

uniqueLang: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#uniqueLang')

update: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#update')

validator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#validator')

value: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#value')

xone: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#xone')

zeroOrMorePath: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#zeroOrMorePath')

zeroOrOnePath: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/shacl#zeroOrOnePath')

class rdflib.SKOS

Bases: *DefinedNamespace*

SKOS Vocabulary

An RDF vocabulary for describing the basic structure and content of concept schemes such as thesauri, classification schemes, subject heading lists, taxonomies, 'folksonomies', other types of controlled vocabulary, and also concept schemes embedded in glossaries and terminologies.

Generated from: https://www.w3.org/2009/08/skos-reference/skos.rdf Date: 2020-05-26 14:20:08.489187

Collection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#Collection')

Concept: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#Concept')

ConceptScheme: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#ConceptScheme')

OrderedCollection: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#OrderedCollection')

altLabel: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#altLabel')

broadMatch: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#broadMatch')

broader: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#broader')

broaderTransitive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#broaderTransitive')

changeNote: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#changeNote')

closeMatch: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#closeMatch')

definition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#definition')

editorialNote: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#editorialNote')

exactMatch: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#exactMatch')

example: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#example')

hasTopConcept: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#hasTopConcept')

hiddenLabel: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#hiddenLabel')

historyNote: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#historyNote')

inScheme: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#inScheme')

mappingRelation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#mappingRelation')

member: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#member')

memberList: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#memberList')

narrowMatch: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#narrowMatch')

narrower: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#narrower')

narrowerTransitive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#narrowerTransitive')

notation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#notation')

note: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#note')

prefLabel: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#prefLabel')

related: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#related')

relatedMatch: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#relatedMatch')

scopeNote: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#scopeNote')

semanticRelation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#semanticRelation')

topConceptOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2004/02/skos/core#topConceptOf')

class rdflib.SOSA

Bases: *DefinedNamespace*

Sensor, Observation, Sample, and Actuator (SOSA) Ontology

This ontology is based on the SSN Ontology by the W3C Semantic Sensor Networks Incubator Group (SSN-XG), together with considerations from the W3C/OGC Spatial Data on the Web Working Group.

Generated from: http://www.w3.org/ns/sosa/ Date: 2020-05-26 14:20:08.792504

ActuatableProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/ActuatableProperty')

Actuation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Actuation')

Actuator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Actuator')

FeatureOfInterest: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/FeatureOfInterest')

ObservableProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/ObservableProperty')

Observation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Observation')

Platform: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Platform')

Procedure: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Procedure')

Result: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Result')

Sample: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Sample')

Sampler: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Sampler')

Sampling: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Sampling')

Sensor: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/Sensor')

actsOnProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/actsOnProperty')

hasFeatureOfInterest: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/hasFeatureOfInterest')

hasResult: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/hasResult')

hasSample: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/hasSample')

hasSimpleResult: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/hasSimpleResult')

hosts: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/hosts')

isActedOnBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/isActedOnBy')

isFeatureOfInterestOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/isFeatureOfInterestOf')

isHostedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/isHostedBy')

isObservedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/isObservedBy')

isResultOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/isResultOf')

isSampleOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/isSampleOf')

madeActuation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/madeActuation')

madeByActuator: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/madeByActuator')

madeBySampler: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/madeBySampler')

madeBySensor: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/madeBySensor')

madeObservation: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/madeObservation')

madeSampling: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/madeSampling')

observedProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/observedProperty')

observes: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/observes')

phenomenonTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/phenomenonTime')

resultTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/resultTime')

usedProcedure: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/sosa/usedProcedure')

class rdflib.**SSN**

Bases: *DefinedNamespace*

Semantic Sensor Network Ontology

This ontology describes sensors, actuators and observations, and related concepts. It does not describe domain concepts, time, locations, etc. these are intended to be included from other ontologies via OWL imports.

Generated from: http://www.w3.org/ns/ssn/ Date: 2020-05-26 14:20:09.068204

Deployment: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/Deployment')

Input: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/Input')

Output: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/Output')

Property: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/Property')

Stimulus: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/Stimulus')

System: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/System')

deployedOnPlatform: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/deployedOnPlatform')

deployedSystem: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/deployedSystem')

detects: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/detects')

forProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/forProperty')

hasDeployment: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/hasDeployment')

hasInput: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/hasInput')

hasOutput: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/hasOutput')

hasProperty: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/hasProperty')

hasSubSystem: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/hasSubSystem')

implementedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/implementedBy')

implements: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/implements')

inDeployment: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/inDeployment')

isPropertyOf: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/isPropertyOf')

isProxyFor: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/isProxyFor')

wasOriginatedBy: *URIRef* = rdflib.term.URIRef('http://www.w3.org/ns/ssn/wasOriginatedBy')

class rdflib.**TIME**

Bases: *DefinedNamespace*

OWL-Time

Generated from: http://www.w3.org/2006/time# Date: 2020-05-26 14:20:10.531265

DateTimeDescription: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#DateTimeDescription')

DateTimeInterval: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#DateTimeInterval')

DayOfWeek: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#DayOfWeek')

Duration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Duration')

DurationDescription: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#DurationDescription')

Friday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Friday')

GeneralDateTimeDescription: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#GeneralDateTimeDescription')

GeneralDurationDescription: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#GeneralDurationDescription')

Instant: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Instant')

Interval: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Interval')

January: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#January')

Monday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Monday')

MonthOfYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#MonthOfYear')

ProperInterval: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#ProperInterval')

Saturday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Saturday')

Sunday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Sunday')

TRS: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TRS')

TemporalDuration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TemporalDuration')

TemporalEntity: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TemporalEntity')

TemporalPosition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TemporalPosition')

TemporalUnit: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TemporalUnit')

Thursday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Thursday')

TimePosition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TimePosition')

TimeZone: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#TimeZone')

Tuesday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Tuesday')

Wednesday: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Wednesday')

Year: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#Year')

after: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#after')

before: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#before')

day: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#day')

dayOfWeek: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#dayOfWeek')

dayOfYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#dayOfYear')

days: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#days')

generalDay: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#generalDay')

generalMonth: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#generalMonth')

generalYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#generalYear')

hasBeginning: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#hasBeginning')

hasDateTimeDescription: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#hasDateTimeDescription')

hasDuration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#hasDuration')

hasDurationDescription: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#hasDurationDescription')

hasEnd: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#hasEnd')

hasTRS: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#hasTRS')

hasTemporalDuration: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#hasTemporalDuration')

hasTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#hasTime')

hasXSDDuration: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#hasXSDDuration')

hour: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#hour')

hours: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#hours')

inDateTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#inDateTime')

inTemporalPosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#inTemporalPosition')

inTimePosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#inTimePosition')

inXSDDate: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#inXSDDate')

inXSDDateTime: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#inXSDDateTime')

inXSDDateTimeStamp: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#inXSDDateTimeStamp')

inXSDgYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#inXSDgYear')

inXSDgYearMonth: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#inXSDgYearMonth')

inside: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#inside')

intervalAfter: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalAfter')

intervalBefore: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalBefore')

intervalContains: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalContains')

intervalDisjoint: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalDisjoint')

intervalDuring: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalDuring')

intervalEquals: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalEquals')

intervalFinishedBy: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalFinishedBy')

intervalFinishes: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalFinishes')

intervalIn: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#intervalIn')

intervalMeets: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalMeets')

intervalMetBy: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalMetBy')

intervalOverlappedBy: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalOverlappedBy')

intervalOverlaps: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalOverlaps')

intervalStartedBy: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalStartedBy')

intervalStarts: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#intervalStarts')

minute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#minute')

minutes: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#minutes')

month: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#month')

monthOfYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#monthOfYear')

months: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#months')

nominalPosition: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2006/time#nominalPosition')

numericDuration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#numericDuration')

numericPosition: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#numericPosition')

second: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#second')

seconds: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#seconds')

timeZone: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#timeZone')

unitDay: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitDay')

unitHour: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitHour')

unitMinute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitMinute')

unitMonth: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitMonth')

unitSecond: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitSecond')

unitType: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitType')

unitWeek: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitWeek')

unitYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#unitYear')

week: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#week')

weeks: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#weeks')

xsdDateTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#xsdDateTime')

year: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#year')

years: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2006/time#years')

class rdflib.**URIRef**(*value: str*, *base: Optional[str] = None*)

Bases: *IdentifiedNode*

RDF 1.1's IRI Section https://www.w3.org/TR/rdf11-concepts/#section-IRIs

---

Note: Documentation on RDF outside of RDFLib uses the term IRI or URI whereas this class is called URIRef. This is because it was made when the first version of the RDF specification was current, and it used the term *URIRef*, see RDF 1.0 URIRef

---

An IRI (Internationalized Resource Identifier) within an RDF graph is a Unicode string that conforms to the syntax defined in RFC 3987.

IRIs in the RDF abstract syntax MUST be absolute, and MAY contain a fragment identifier.

IRIs are a generalization of URIs [RFC3986] that permits a wider range of Unicode characters.

__add__(*other*)

Return self+value.

Return type
URIRef

__annotations__ = {'__invert__': typing.Callable[[ForwardRef('URIRef')], ForwardRef('InvPath')], '__neg__': typing.Callable[[ForwardRef('URIRef')], ForwardRef('NegatedPath')], '__or__': typing.Callable[[ForwardRef('URIRef'), typing.Union[ForwardRef('URIRef'), ForwardRef('Path')]], ForwardRef('AlternativePath')], '__truediv__': typing.Callable[[ForwardRef('URIRef'), typing.Union[ForwardRef('URIRef'), ForwardRef('Path')]], ForwardRef('SequencePath')]}

__invert__()

inverse path

Parameters
p (Union[*URIRef*, *Path*]) –

Return type
InvPath

__mod__(*other*)

Return self%value.

Return type
URIRef

__module__ = 'rdflib.term'

__mul__(*mul*)

cardinality path

Parameters

- **p** (Union[*URIRef*, *Path*]) –
- **mul** (Literal['*', '+', '?']) –

Return type
MulPath

__neg__()

negated path

Parameters
p (Union[*URIRef*, *AlternativePath*, *InvPath*]) –

Return type
NegatedPath

static __new__(*cls*, *value*, *base=None*)

Parameters

- **value** (str) –
- **base** (Optional[str]) –

Return type
URIRef

__or__(*other*)

alternative path

Parameters

- **self** (Union[*URIRef*, *Path*]) –

- **other** (Union[*URIRef*, *Path*]) –

__radd__(*other*)

Return type
URIRef

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*URIRef*], Tuple[str]]

__repr__()

Return repr(self).

Return type
str

__slots__ = ()

__truediv__(*other*)

sequence path

Parameters

- **self** (Union[*URIRef*, *Path*]) –
- **other** (Union[*URIRef*, *Path*]) –

de_skolemize()

Create a Blank Node from a skolem URI, in accordance with http://www.w3.org/TR/rdf11-concepts/#section-skolemization. This function accepts only rdflib type skolemization, to provide a round-tripping within the system.

New in version 4.0.

Return type
BNode

defrag()

Return type
URIRef

property fragment: str

Return the URL Fragment

```
>>> URIRef("http://example.com/some/path/#some-fragment").fragment
'some-fragment'
>>> URIRef("http://example.com/some/path/").fragment
''
```

Return type
str

n3(*namespace_manager=None*)

This will do a limited check for valid URIs, essentially just making sure that the string includes no illegal characters (<, >, ", {, }, |, \, `, ^)

Parameters
namespace_manager (Optional[*NamespaceManager*]) – if not None, will be used to make up a prefixed name

Return type
str

class rdflib.**VANN**

Bases: *DefinedNamespace*

VANN: A vocabulary for annotating vocabulary descriptions

This document describes a vocabulary for annotating descriptions of vocabularies with examples and usage notes.

Generated from: https://vocab.org/vann/vann-vocab-20100607.rdf Date: 2020-05-26 14:21:15.580430

changes: *URIRef* = rdflib.term.URIRef('http://purl.org/vocab/vann/changes')

example: *URIRef* = rdflib.term.URIRef('http://purl.org/vocab/vann/example')

preferredNamespacePrefix: *URIRef* = rdflib.term.URIRef('http://purl.org/vocab/vann/preferredNamespacePrefix')

preferredNamespaceUri: *URIRef* = rdflib.term.URIRef('http://purl.org/vocab/vann/preferredNamespaceUri')

termGroup: *URIRef* = rdflib.term.URIRef('http://purl.org/vocab/vann/termGroup')

usageNote: *URIRef* = rdflib.term.URIRef('http://purl.org/vocab/vann/usageNote')

class rdflib.**VOID**

Bases: *DefinedNamespace*

Vocabulary of Interlinked Datasets (VoID)

The Vocabulary of Interlinked Datasets (VoID) is an RDF Schema vocabulary for expressing metadata about RDF datasets. It is intended as a bridge between the publishers and users of RDF data, with applications ranging from data discovery to cataloging and archiving of datasets. This document provides a formal definition of the new RDF classes and properties introduced for VoID. It is a companion to the main specification document for VoID, Describing Linked Datasets with the VoID Vocabulary.

Generated from: http://rdfs.org/ns/void# Date: 2020-05-26 14:20:11.911298

Dataset: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#Dataset')

DatasetDescription: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#DatasetDescription')

Linkset: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#Linkset')

TechnicalFeature: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#TechnicalFeature')

classPartition: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#classPartition')

classes: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#classes')

dataDump: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#dataDump')

distinctObjects: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#distinctObjects')

distinctSubjects: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#distinctSubjects')

documents: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#documents')

entities: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#entities')

exampleResource: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#exampleResource')

feature: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#feature')

inDataset: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#inDataset')

linkPredicate: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#linkPredicate')

objectsTarget: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#objectsTarget')

openSearchDescription: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#openSearchDescription')

properties: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#properties')

property: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#property')

propertyPartition: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#propertyPartition')

rootResource: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#rootResource')

sparqlEndpoint: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#sparqlEndpoint')

subjectsTarget: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#subjectsTarget')

subset: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#subset')

target: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#target')

triples: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#triples')

uriLookupEndpoint: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#uriLookupEndpoint')

uriRegexPattern: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#uriRegexPattern')

uriSpace: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#uriSpace')

vocabulary: *URIRef* = rdflib.term.URIRef('http://rdfs.org/ns/void#vocabulary')

class rdflib.**Variable**(*value: str*)

Bases: *Identifier*

A Variable - this is used for querying, or in Formula aware graphs, where Variables can be stored

__annotations__ = {}

__module__ = 'rdflib.term'

static __new__(*cls*, *value*)

Parameters
value (str) –

Return type
Variable

__reduce__()

Helper for pickle.

Return type
Tuple[Type[*Variable*], Tuple[str]]

__repr__()

Return repr(self).

Return type
str

__slots__ = ()

n3(*namespace_manager=None*)

Parameters
namespace_manager (Optional[*NamespaceManager*]) –

Return type
str

toPython()

Return type
str

class rdflib.**XSD**

Bases: *DefinedNamespace*

W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes

Generated from: ../schemas/datatypes.xsd Date: 2021-09-05 20:37+10

Assertions: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#Assertions')

ENTITIES: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#ENTITIES')

ENTITY: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#ENTITY')

ID: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#ID')

IDREF: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#IDREF')

IDREFS: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#IDREFS')

NCName: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#NCName')

NMTOKEN: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#NMTOKEN')

NMTOKENS: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#NMTOKENS')

NOTATION: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#NOTATION')

Name: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#Name')

QName: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#QName')

anyURI: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#anyURI')

base64Binary: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#base64Binary')

boolean: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#boolean')

bounded: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#bounded')

byte: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#byte')

cardinality: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#cardinality')

date: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#date')

dateTime: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dateTime')

dateTimeStamp: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dateTimeStamp')

day: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#day')

dayTimeDuration: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#dayTimeDuration')

decimal: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#decimal')

double: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#double')

duration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#duration')

enumeration: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#enumeration')

explicitTimezone: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#explicitTimezone')

float: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#float')

fractionDigits: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#fractionDigits')

gDay: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#gDay')

gMonth: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#gMonth')

gMonthDay: *URIRef* =
rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#gMonthDay')

gYear: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#gYear')

gYearMonth: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#gYearMonth')

hexBinary: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#hexBinary')

hour: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#hour')

int: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#int')

integer: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#integer')

language: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#language')

length: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#length')

long: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#long')

maxExclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#maxExclusive')

maxInclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#maxInclusive')

maxLength: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#maxLength')

minExclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#minExclusive')

minInclusive: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#minInclusive')

minLength: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#minLength')

minute: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#minute')

month: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#month')

negativeInteger: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#negativeInteger')

nonNegativeInteger: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#nonNegativeInteger')

nonPositiveInteger: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#nonPositiveInteger')

normalizedString: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#normalizedString')

numeric: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#numeric')

ordered: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#ordered')

pattern: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#pattern')

positiveInteger: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#positiveInteger')

second: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#second')

short: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#short')

string: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#string')

time: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#time')

timezoneOffset: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#timezoneOffset')

token: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#token')

totalDigits: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#totalDigits')

unsignedByte: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#unsignedByte')

unsignedInt: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#unsignedInt')

unsignedLong: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#unsignedLong')

unsignedShort: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#unsignedShort')

whiteSpace: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#whiteSpace')

year: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#year')

yearMonthDuration: *URIRef* = rdflib.term.URIRef('http://www.w3.org/2001/XMLSchema#yearMonthDuration')

3.2 Plugins

Many parts of RDFLib are extensible with plugins, see setuptools' 'Creating and discovering plugins'. These pages list the plugins included in RDFLib core.

3.2.1 Plugin parsers

These serializers are available in default RDFLib, you can use them by passing the name to graph's *parse()* method:

```
graph.parse(my_url, format='n3')
```

The `html` parser will auto-detect RDFa, HTurtle or Microdata.

It is also possible to pass a mime-type for the `format` parameter:

```
graph.parse(my_url, format='application/rdf+xml')
```

If you are not sure what format your file will be, you can use *rdflib.util.guess_format()* which will guess based on the file extension.

Name	Class
json-ld	*JsonLDParser*
hext	*HextuplesParser*
n3	*N3Parser*
nquads	*NQuadsParser*
nt	*NTParser*
trix	*TriXParser*
turtle	*TurtleParser*
xml	*RDFXMLParser*

Multi-graph IDs

Note that for correct parsing of multi-graph data, e.g. Trig, HexT, etc., into a `ConjunctiveGraph` or a `Dataset`, as opposed to a context-unaware `Graph`, you will need to set the `publicID` of the `ConjunctiveGraph` a `Dataset` to the identifier of the `default_context` (default graph), for example:

```
d = Dataset()
d.parse(
    data=""" ... """,
    format="trig",
    publicID=d.default_context.identifier
)
```

(from the file tests/test_serializer_hext.py)

3.2.2 Plugin serializers

These serializers are available in default RDFLib, you can use them by passing the name to a graph's *serialize()* method:

```
print graph.serialize(format='n3')
```

It is also possible to pass a mime-type for the `format` parameter:

```
graph.serialize(my_url, format='application/rdf+xml')
```

Name	Class
json-ld	*JsonLDSerializer*
n3	*N3Serializer*
nquads	*NQuadsSerializer*
nt	*NTSerializer*
hext	*HextuplesSerializer*
pretty-xml	*PrettyXMLSerializer*
trig	*TrigSerializer*
trix	*TriXSerializer*
turtle	*TurtleSerializer*
longturtle	*LongTurtleSerializer*
xml	*XMLSerializer*

JSON-LD

JSON-LD - 'json-ld' - has been incorprated in rdflib since v6.0.0.

HexTuples

The HexTuples Serializer - 'hext' - uses the HexTuples format defined at https://github.com/ontola/hextuples.

For serialization of non-context-aware data sources, e.g. a single `Graph`, the 'graph' field (6th variable in the Hextuple) will be an empty string.

For context-aware (multi-graph) serialization, the 'graph' field of the default graph will be an empty string and the values for other graphs will be Blank Node IDs or IRIs.

3.2.3 Plugin stores

Built In

The following Stores are contained within the rdflib core package:

Name	Class
Auditable	*AuditableStore*
Concurrent	*ConcurrentStore*
SimpleMemory	*SimpleMemory*
Memory	*Memory*
SPARQLStore	*SPARQLStore*
SPARQLUpdateStore	*SPARQLUpdateStore*
BerkeleyDB	*BerkeleyDB*
default	*Memory*

External

The following Stores are defined externally to rdflib's core package, so look to their documentation elsewhere for specific details of use.

Name	Repository	Notes
SQLAlchemy	https://github.com/RDFLib/rdflib-sqlalchemy	An SQLAlchemy-backed, formula-aware RDFLib Store. Tested dialects are: SQLite, MySQL & PostgreSQL
leveldb	https://github.com/RDFLib/rdflib-leveldb	An adaptation of RDFLib BerkeleyDB Store's key-value approach, using LevelDB as a back-end
Kyoto Cabinet	https://github.com/RDFLib/rdflib-kyotocabinet	An adaptation of RDFLib BerkeleyDB Store's key-value approach, using Kyoto Cabinet as a back-end
HDT	https://github.com/RDFLib/rdflib-hdt	A Store back-end for rdflib to allow for reading and querying HDT documents
Oxigraph	https://github.com/oxigraph/oxrdflib	Works with the Pyoxigraph Python graph database library

If you have, or know of a Store implementation and would like it listed here, please submit a Pull Request!

Use

You can use these stores like this:

```
from rdflib import Graph

# use the default memory Store
graph = Graph()

# use the BerkeleyDB Store
graph = Graph(store="BerkeleyDB")
```

In some cases, you must explicitly *open* and *close* a store, for example:

```
from rdflib import Graph

# use the BerkeleyDB Store
graph = Graph(store="BerkeleyDB")
graph.open("/some/folder/location")
# do things ...
graph.close()
```

3.2.4 Plugin query results

Plugins for reading and writing of (SPARQL) *Result* - pass name to either *parse()* or *serialize()*

Parsers

Name	Class
csv	*CSVResultParser*
json	*JSONResultParser*
tsv	*TSVResultParser*
xml	*XMLResultParser*

Serializers

Name	Class
csv	*CSVResultSerializer*
json	*JSONResultSerializer*
txt	*TXTResultSerializer*
xml	*XMLResultSerializer*

CHAPTER

FOUR

FOR DEVELOPERS

4.1 RDFLib developers guide

4.1.1 Introduction

This document describes the process and conventions to follow when developing RDFLib code.

- Please be as Pythonic as possible (**PEP 8**).
- Code should be formatted using black and we use Black v23.1.0, with the black config in `pyproject.toml`.
- Code should also pass flake8 linting and mypy type checking.
- You must supply tests for new code.
- RDFLib uses Poetry for dependency management and packaging.

If you add a new cool feature, consider also adding an example in `./examples`

4.1.2 Pull Requests Guidelines

Contributions to RDFLib are made through pull requests (PRs).

In general, maintainers will only merge PRs if the following conditions are met:

- The PR has been sufficiently reviewed.

  Each PR should be reviewed and approved by at least two people other than the author of the PR before it is merged and PRs will be processed faster if they are easier to review and approve of.

  Reviews are open to everyone, but the weight assigned to any particular review is at the discretion of maintainers.

- Changes that have a runtime impact are covered by unit tests.

  There should either be existing tests that cover the changed code and behaviour, or the PR should include tests. For more information about what is considered adequate testing see the *Tests section*.

- Documentation that covers something that changed has been updated.
- Type checks and unit tests that are part of our continuous integration workflow pass.

In addition to these conditions, PRs that are easier to review and approve will be processed quicker. The primary factors that determine this is the scope and size of a PR. If there are few changes and the scope is limited then there is less that a reviewer has to understand and less that they can disagree with. It is thus important to try and split up your changes into multiple independent PRs if possible. No PR is too small.

For PRs that introduce breaking changes, it is even more critical that they are limited in size and scope, as they will likely have to be kept up to date with the `main` branch of this project for some time before they are merged.

It is also critical that your PR is understandable both in what it does and why it does it, and how the change will impact the users of this project, for this reason it is essential that your PR's description explains the nature of the PR, what the PR intends to do, why this is desirable, and how this will affect the users of this project.

Please note that while we would like all PRs to follow the guidelines given here, we will not reject a PR just because it does not.

4.1.3 Tests

Any new functionality being added to RDFLib *must* have unit tests and should have doc tests supplied.

Typically, you should add your functionality and new tests to a branch of RDFlib and and run all tests locally and see them pass. There are currently close to 4,000 tests with a few extra expected failures and skipped tests. We won't allow Pull Requests that break any of the existing tests.

Tests that you add should show how your new feature or bug fix is doing what you say it is doing: if you remove your enhancement, your new tests should fail!

Finally, please consider adding simple and more complex tests. It's good to see the basic functionality of your feature tests and then also any tricky bits or edge cases.

Testing framework

RDFLib uses the pytest testing framework.

Running tests

To run RDFLib's test suite with pytest:

```
$ poetry install
$ poetry run pytest
```

Specific tests can be run by file name. For example:

```
$ poetry run pytest test/test_graph/test_graph.py
```

For more extensive tests, including tests for the berkleydb backend, install extra requirements before executing the tests.

```
$ poetry install --all-extras
$ poetry run pytest
```

Writing tests

New tests should be written for pytest instead of for python's built-in `unittest` module as pytest provides advanced features such as parameterization and more flexibility in writing expected failure tests than `unittest`.

A primer on how to write tests for pytest can be found here.

The existing tests that use `unittest` work well with pytest, but they should ideally be updated to the pytest test-style when they are touched.

Test should go into the `test/` directory, either into an existing test file with a name that is applicable to the test being written, or into a new test file with a name that is descriptive of the tests placed in it. Test files should be named `test_*.py` so that pytest can discover them.

4.1.4 Running static checks

Check formatting with black, making sure you use our black.toml config file:

```
poetry run black .
```

Check style and conventions with flake8:

```
poetry run flake8 rdflib
```

We also provide a flakeheaven baseline that ignores existing flake8 errors and only reports on newly introduced flake8 errors:

```
poetry run flakeheaven
```

Check types with mypy:

```
poetry run mypy --show-error-context --show-error-codes
```

4.1.5 pre-commit and pre-commit ci

We have pre-commit configured with black for formatting code.

Some useful commands for using pre-commit:

```
# Install pre-commit.
pip install --user --upgrade pre-commit

# Install pre-commit hooks, this will run pre-commit
# every time you make a git commit.
pre-commit install

# Run pre-commit on changed files.
pre-commit run

# Run pre-commit on all files.
pre-commit run --all-files
```

There is also two tox environments for pre-commit:

```
# run pre-commit on changed files.
tox -e precommit

# run pre-commit on all files.
tox -e precommitall
```

There is no hard requirement for pull requests to be processed with pre-commit (or the underlying processors), however doing this makes for a less noisy codebase with cleaner history.

We have enabled https://pre-commit.ci/ and this can be used to automatically fix pull requests by commenting `pre-commit.ci autofix` on a pull request.

4.1.6 Using tox

RDFLib has a tox config file that makes it easier to run validation on all supported python versions.

```
# Install tox.
pip install tox

# List the tox environments that run by default.
tox -e

# Run the default environments.
tox

# List all tox environments, including ones that don't run by default.
tox -a

# Run a specific environment.
tox -e py37 # default environment with py37
tox -e py39-extra # extra tests with py39

# Override the test command.
# the below command will run `pytest test/test_translate_algebra.py`
# instead of the default pytest command.
tox -e py37,py39 -- pytest test/test_translate_algebra.py
```

4.1.7 `go-task` and `Taskfile.yml`

A `Taskfile.yml` is provided for go-task with various commands that facilitate development.

Instructions for installing go-task can be seen in the go-task installation guide.

Some useful commands for working with the task in the taskfile is given below:

```
# List available tasks.
task -l

# Configure the environment for development
task configure

# Run basic validation
task validate

# Build docs
task docs:build

# Run live-preview on the docs
task docs:live-server

# Run the py310 tox environment
task tox -- -e py310
```

The Taskfile usage documentation provides more information on how to work with taskfiles.

4.1.8 Development container

To simplify the process of getting a working development environment to develop rdflib in we provide a Development Container (*devcontainer*) that is configured in Docker Compose. This container can be used directly to run various commands, or it can be used with editors that support Development Containers.

Important: The devcontainer is intended to run with a rootless docker daemon so it can edit files owned by the invoking user without an invovled configuration process.

Using a rootless docker daemon also has general security benefits.

To use the development container directly:

```
# Build the devcontainer docker image.
docker-compose build

# Configure the system for development.
docker-compose run --rm run task configure

# Run the validate task inside the devtools container.
docker-compose run --rm run task validate

# Run extensive tests inside the devtools container.
docker-compose run --rm run task EXTENSIVE=true test

# To get a shell into the devcontainer docker image.
docker-compose run --rm run bash
```

The devcontainer also works with Podman Compose.

Details on how to use the development container with VSCode can found in the Developing inside a Container page. With the VSCode development container CLI installed the following command can be used to open the repository inside the development container:

```
# Inside the repository base directory
cd ./rdflib/

# Build the development container.
devcontainer build .

# Open the code inside the development container.
devcontainer open .
```

4.1.9 Writing documentation

We use sphinx for generating HTML docs, see *Writing RDFLib Documentation*.

4.1.10 Continuous Integration

We used GitHub Actions for CI, see:

https://github.com/RDFLib/rdflib/actions

If you make a pull-request to RDFLib on GitHub, GitHub Actions will automatically test your code and we will only merge code passing all tests.

Please do *not* commit tests you know will fail, even if you're just pointing out a bug. If you commit such tests, flag them as expecting to fail.

4.1.11 Compatibility

RDFlib 6.0.0 release and later only support Python 3.7 and newer.

RDFLib 5.0.0 maintained compatibility with Python versions 2.7, 3.4, 3.5, 3.6, 3.7.

4.1.12 Releasing

Set to-be-released version number in `rdflib/__init__.py` and `README.md`. Check date in `LICENSE`.

Add `CHANGELOG.md` entry.

Commit this change. It's preferable make the release tag via https://github.com/RDFLib/rdflib/releases/new :: Our Tag versions aren't started with 'v', so just use a plain 5.0.0 like version. Release title is like "RDFLib 5.0.0", the description a copy of your `CHANGELOG.md` entry. This gives us a nice release page like this:: https://github.com/RDFLib/rdflib/releases/tag/4.2.2

If for whatever reason you don't want to take this approach, the old one is:

```
Tagging the release commit with::

  git tag -am 'tagged version' X.X.X

When pushing, remember to do::

  git push --tags
```

No matter how you create the release tag, remember to upload tarball to pypi with:

```
rm -r dist/X.X.X[.-]*  # delete all previous builds for this release, just in case

rm -r build
python setup.py sdist
python setup.py bdist_wheel
ls dist

# upload with twine
# WARNING: once uploaded can never be modified, only deleted!
twine upload dist/rdflib-X.X.X[.-]*
```

Set new dev version number in the above locations, i.e. next release `-dev`: `5.0.1-dev` and commit again.

Tweet, email mailing list and inform members in the chat.

4.2 Contributor Covenant Code of Conduct

4.2.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

4.2.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.2.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

4.2.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

4.2.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at https://github.com/RDFLib/rdflib/discussions. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

4.2.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

4.2.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

4.3 Writing RDFLib Documentation

These docs are generated with Sphinx.

Sphinx makes it very easy to pull in doc-strings from modules, classes, methods, etc. When writing doc-strings, special reST fields can be used to annotate parameters, return-types, etc. This makes for pretty API docs. See here for the Shinx documentation about these fields.

4.3.1 Building

To build the documentation you can use Sphinx from within the poetry environment. To do this, run the following commands:

```
# Install poetry venv
poetry install

# Build the sphinx docs
poetry run sphinx-build -b html -d docs/_build/doctrees docs docs/_build/html
```

Docs will be generated in `docs/_build/html` and API documentation, generated from doc-strings, will be placed in `docs/apidocs/`.

There is also a tox environment for building documentation:

```
tox -e docs
```

4.3.2 API Docs

API Docs are automatically generated with `sphinx-apidoc`:

```
poetry run sphinx-apidoc -f -d 10 -o docs/apidocs/ rdflib examples
```

Note that `rdflib.rst` was manually tweaked so as to not include all
imports in `rdflib/__init__.py`.

4.3.3 Tables

The tables in `plugin_*.rst` were generated with `plugintable.py`

4.4 Persisting Notation 3 Terms

4.4.1 Using N3 Syntax for Persistence

Blank Nodes, Literals, URI References, and Variables can be distinguished in persistence by relying on Notation 3 syntax convention.

All URI References can be expanded and persisted as:

```
<..URI..>
```

All Literals can be expanded and persisted as:

```
"..value.."@lang or "..value.."^^dtype_uri
```

Note: `@lang` is a language tag and `^^dtype_uri` is the URI of a data type associated with the Literal

---

Blank Nodes can be expanded and persisted as:

```
_:Id
```

Note: where Id is an identifier as determined by skolemization. Skolemization is a syntactic transformation routinely used in automatic inference systems in which existential variables are replaced by 'new' functions - function names not used elsewhere - applied to any enclosing universal variables. In RDF, Skolemization amounts to replacing every blank node in a graph by a 'new' name, i.e. a URI reference which is guaranteed to not occur anywhere else. In effect, it gives 'arbitrary' names to the anonymous entities whose existence was asserted by the use of blank nodes: the arbitrariness of the names ensures that nothing can be inferred that would not follow from the bare assertion of existence represented by the blank node. (Using a literal would not do. Literals are never 'new' in the required sense.)

Variables can be persisted as they appear in their serialization `(?varName)` - since they only need be unique within their scope (the context of their associated statements)

These syntactic conventions can facilitate term round-tripping.

4.4.2 Variables by Scope

Would an interface be needed in order to facilitate a quick way to aggregate all the variables in a scope (given by a formula identifier)? An interface such as:

```
def variables(formula_identifier)
```

4.4.3 The Need to Skolemize Formula Identifiers

It would seem reasonable to assume that a formula-aware store would assign Blank Node identifiers as names of formulae that appear in a N3 serialization. So for instance, the following bit of N3:

```
{?x a :N3Programmer} => {?x :has :Migrane}
```

Could be interpreted as the assertion of the following statement:

```
_:a log:implies _:b
```

However, how are `_:a` and `_:b` distinguished from other Blank Nodes? A formula-aware store would be expected to persist the first set of statements as quoted statements in a formula named `_:a` and the second set as quoted statements in a formula named `_:b`, but it would not be cost-effective for a serializer to have to query the store for all statements in a context named `_:a` in order to determine if `_:a` was associated with a formula (so that it could be serialized properly).

4.4.4 Relying on `log:Formula` Membership

The store could rely on explicit `log:Formula` membership (via `rdf:type` statements) to model the distinction of Blank Nodes associated with formulae. However, would these statements be expected from an N3 parser or known implicitly by the store? i.e., would all such Blank Nodes match the following pattern:

```
?formula rdf:type log:Formula
```

4.4.5 Relying on an Explicit Interface

A formula-aware store could also support the persistence of this distinction by implementing a method that returns an iterator over all the formulae in the store:

```
def formulae(triple=None)
```

This function would return all the Blank Node identifiers assigned to formulae or just those that contain statements matching the given triple pattern and would be the way a serializer determines if a term refers to a formula (in order to properly serializer it).

How much would such an interface reduce the need to model formulae terms as first class objects (perhaps to be returned by the *triples()* function)? Would it be more useful for the *Graph* (or the store itself) to return a Context object in place of a formula term (using the formulae interface to make this determination)?

Conversely, would these interfaces (variables and formulae) be considered optimizations only since you have the distinction by the kinds of terms triples returns (which would be expanded to include variables and formulae)?

4.4.6 Persisting Formula Identifiers

This is the most straight forward way to maintain this distinction - without relying on extra interfaces. Formula identifiers could be persisted distinctly from other terms by using the following notation:

```
{_:bnode} or {<.. URI ..>}
```

This would facilitate their persistence round-trip - same as the other terms that rely on N3 syntax to distinguish between each other.

4.5 Type Hints

This document provides some details about the type hints for RDFLib. More information about type hints can be found here

4.5.1 Rationale for Type Hints

Type hints are code annotations that describe the types of variables, function parameters and function return value types in a way that can be understood by humans, static type checkers like mypy, code editors like VSCode, documentation generators like Sphinx, and other tools.

Static type checkers can use type hints to detect certain classes of errors by inspection. Code editors and IDEs can use type hints to provide better auto-completion and documentation generators can use type hints to generate better documentation.

These capabilities make it easier to develop a defect-free RDFLib and they also make it easier for users of RDFLib who can now use static type checkers to detect type errors in code that uses RDFLib.

4.5.2 Gradual Typing Process

Type hints are being added to RDFLib through a process called gradual typing. This process involves adding type hints to some parts of RDFLib while leaving the rest without type hints. Gradual typing is being applied to many, long-lived, Python code bases.

This process is beneficial in that we can realize some of the benefits of type hints without requiring that the whole codebase have type hints.

4.5.3 Intended Type Hints

The intent is to have type hints in place for all of RDFLib and to have these type hints be as accurate as possible.

The accuracy of type hints is determined by both the standards that RDFLib aims to conform to, like RDF 1.1, and the deliberate choices that are made when implementing RDFLib. For example, given that the RDF 1.1 specification stipulates that the subject of an RDF triple cannot be a literal, all functions that accept an *RDF term* to be used as the subject of a triple should have type hints which excludes values that are literals.

There may be cases where some functionality of RDFLib may work perfectly well with values of types that are excluded by the type hints, but if these additional types violate the relevant standards we will consider the correct type hints to be those that exclude values of these types.

4.5.4 Public Type Aliases

In python, type hints are specified in annotations. Type hints are different from type aliases which are normal python variables that are not intended to provide runtime utility and are instead intended for use in static type checking.

For clarity, the following is an example of a function `foo` with type hints:

```
def foo(a: int) -> int:
    return a + 1
```

In the function `foo`, the input variable `a` is indicated to be of type `int` and the function is indicated to return an `int`.

The following is an example of a type alias `Bar`:

```
from typing import Tuple

Bar = Tuple[int, str]
```

RDFLib will provide public type aliases under the `rdflib.typing` package, for example, `rdflib.typing.Triple`, `rdflib.typing.Quad`. Type aliases in the rest of RDFLib should be private (i.e. being with an underscore).

4.5.5 Versioning, Compatibility and Stability

RDFLib attempts to adhere to semver 2.0 which is concerned with the public API of software.

Ignoring type hints, the public API of RDFLib exists implicitly as a consequence of the code of RDFLib and the actual behaviour this entails, the relevant standards that RDFLib is trying to implement, and the documentation of RDFLib, with some interplay between all three of these. RDFLib's public API includes public type aliases, as these are normal python variables and not annotations.

Type hints attempt to formally document RDFLib's implicitly-defined public API in a machine-readable fashion as accurately and correctly as possible within the framework outline earlier in this document.

Type hints do not affect the runtime API or behaviour of RDFLib. In this way then, they are somewhat outside of the scope of semver, however, they still have an impact on the users of RDFLib, even if this impact is not at runtime, but during development. This necessitates some clarity as to what users of RDFLib should expect regarding type hints in RDFLib releases.

Changes to type hints can broadly be classified as follow:

Type Declaration
: Adding type hints to existing code that had no explicit type hints, for example, changing

```
def foo(val):
    return val + 1
```

to

```
def foo(val: int) -> int:
    return val + 1
```

Type Refinement
: Refining existing type hints to be narrower, for example, changing a type hint of `typing.Collection` to `typing.Sequence`.

Type Corrections
: Correcting existing type hints which contradict the behaviour of the code or relevant specifications, for example, changing `typing.Sequence` from `typing.Set`

Given semver version components `MAJOR.MINOR.PATCH`, RDFLib will attempt to constrain type hint changes as follow:

Version Component	Type Declaration	Type Refinement	Type Corrections
MAJOR	YES	YES	YES
MINOR	YES	YES	YES
PATCH	NO	NO	YES

Caution: A caveat worth nothing here is that code that passed type validation on one version of RDFLib can fail type validation on a later version of RDFLib that only differs in `PATCH` version component. This is as a consequence of potential *Type Corrections*.

4.6 RDFLib Contributing Guide

Thank you for considering contributing to RDFLib. This project has no formal funding or full-time maintainers and relies entirely on independent contributors to keep it alive and relevant.

4.6.1 Ways to contribute

Some ways in which you can contribute to RDFLib are:

- Address open issues:
- Fix expected failure tests.
- Add additional expected failure tests for open issues:
- Add tests for untested code:
- Review pull requests marked with the
- Answer questions on Stack Overflow:
- Convert `unittest` based tests to `pytest` based tests:
- Add, correct or improve docstrings:
- Update the RDFLib Wikipedia entry:
- Update the RDFLib Wikidata entry:
- Participate on Gitter/Matrix chat:
- Participate in GitHub discussions:
- Fix flake8 failures.

4.6.2 Pull Requests

Contributions that involve changes to the RDFLib repository have to be made with pull requests and should follow the *RDFLib developers guide*.

4.6.3 Code of Conduct

All contributions to the project should be consistent with the *code of conduct* adopted by RDFLib.

4.7 Decision Records

To ensure that significant changes to RDFLib are made with sufficient consultation, consideration and planning they should be preceded by a decision record that captures the particulars of the decision that lead to the change.

Decision records present the users and maintainers of RDFLib with an opportunity to review decisions before effort is expended to implement the decision in code, and it also makes it possible to review decisions without having to reconstruct them from the code changes that implement them.

Whether a change is significant is hard to measure objectively, but some characteristics that may indicate that a change is significant include:

- It will require changes to code that use RDFLib.
- It cannot be reversed without requiring changes to code that use RDFLib.
- It is onerous to reverse later.
- It increases the maintenance burden of RDFLib.
- It is very large.

Some of these characteristics are not binary but measured in degrees, so some discretion is required when determining if an architectural decision record is appropriate.

Decision records may also be used for changes that do not have any of the listed characteristics if a decision record would be otherwise helpful, for example to capture a decision to change the maintenance process of RDFLib.

Changes not preceded by decision records won't be rejected solely on this basis even if they are deemed significant, and decision records may also be created retrospectively for changes.

Decision records as described here are similar to the concept of Architectural Decision Records, though it is slightly broader as it could include decisions which are not classified as architectural.

4.7.1 Creating a decision record

Decision records should be added to the RDFLib repository in the `./docs/decisions/` directory with a name `{YYYYmmdd}-{title}.rst`.

The content of the decision record should succinctly describe the context of the decision, the decision itself, and the status of the decision.

Decision records should preferably follow Michael Nygard decision record template that he described in a 2011 article on documenting architecture decisions.

For questions about decision records please reach out to the RDFLib maintainers and community using the options given in *Further help & Contact*.

4.7.2 Decision list

Default Branch Name

Status

Accepted

Context

In recent years usage of the word `master` has become somewhat controversial [SFC-BNAMING] and consequently default branch name of Git repos has become `main`, both in Git itself [SFC-BNAMING] and in Git hosting solutions such as GitHub [GH-BRANCHES].

Decision

RDFLib's default branch will be renamed from `master` to `main`. This is primarily to stay in line with modern conventions and to adhere to the principle of least surprise.

Consequences

Anticipated negative consequences:

- Some links to old code will be broken.
- Some people's workflow may break unexpectedly and need adjusting.
- Any code and systems reliant on the old default branch name will fail.

Anticipated positive consequences:

- It will become a bit easier to work with RDFLib for developers that are used to `main` as the default branch.

References

CHAPTER

FIVE

SOURCE CODE

The rdflib source code is hosted on GitHub at https://github.com/RDFLib/rdflib where you can lodge Issues and create Pull Requests to help improve this community project!

The RDFlib organisation on GitHub at https://github.com/RDFLib maintains this package and a number of other RDF and RDFlib-related packaged that you might also find useful.

CHAPTER

SIX

FURTHER HELP & CONTACT

If you would like help with using RDFlib, rather than developing it, please post a question on StackOverflow using the tag `[rdflib]`. A list of existing `[rdflib]` tagged questions can be found here.

You might also like to join RDFlib's dev mailing list or use RDFLib's GitHub discussions section.

The chat is available at gitter or via matrix #RDFLib_rdflib:gitter.im.

BIBLIOGRAPHY

[GH-BRANCHES] GitHub: About the default branch

[SFC-BNAMING] Regarding Git and Branch Naming

PYTHON MODULE INDEX

e

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z