OldMan Documentation

Release 0.1

Benjamin Cogrel

December 01, 2014

Contents

OldMan is a Python *Object Linked Data Mapper* (OLDM). It relies on the popular RDFlib Python library. See the *foreword* for further characteristics.

CHAPTER 1

User's Guide

1.1 Foreword

OldMan is a Python *Object Linked Data Mapper* (OLDM).

An OLDM let you create, retrieve and update RDF representations of Web Resources by manipulating them as Python objects.

OldMan, in its core, is based on two W3C standards:

1. RDF (the Resource Description Framework) as data model;
2. JSON-LD context for mapping objects and RDF graphs.

It is designed to support multiple protocols for interacting with data stores hosting these resources. Currently, only SPARQL is officially supported.

OldMan relies on the RDFlib Python library.

1.1.1 Why a new term?

Some similar projects employ the term *Object RDF Mapper* for denoting the mapping between objects and **RDF graphs**. This terminology uses the same initials than the well-known notion of *Object Relational Mapper* (ORM) that consider *table rows* instead of *RDF graphs*.

The *Object Linked Data Mapper* (OLDM) term avoids this confusion. It also emphasizes that the manipulated resources are supposed to be **on the Web**, not just in a local database. It should lead users to interact with data stores on which they not always have full control (e.g. a tiers Web API).

1.1.2 Mission

OldMan has one main objective: help you to **declare your models using RDF triples and JSON-LD contexts** instead of programming Python model classes yourself.

However, OldMan does not force you to express all your domain logic in a declarative style. OldMan makes easy for you to add dynamically plain-old Python methods to resource objects.

By adopting a declarative style:

1. You can provide both RDF and JSON data to your clients.
2. Your schema (including validation constraints) can be published and reused by **hypermedia-driven** Web clients.
3. Your declared domain logic becomes independent of Python and its frameworks.

It also acknowledges that IRIs or compact URIs (CURIEs) -like strings are not always pleasant to use: arbitrary short names and objects are usually more user-friendly. However, you can still manipulate IRIs when it is relevant for you to do so. Everything remains mapped to IRIs.

1.1.3 Current core features

- Resource-centric validation based on RDF vocabularies:
  - Hydra: hydra:required , hydra:readonly and hydra:writeonly;
  - Literal validation for common XSD types;
  - Literal validation for arbitrary property (e.g. foaf:mbox);
  - JSON-LD collections (set, list and language maps);
- IRI generation for new resources (objects);
- Inheritance (attributes and Python methods);
- An attribute can require its value to be a collection (a set, a list or a language map);
- Arbitrary attribute names (e.g. plural names for collections);
- Extensibility to various sorts of data stores (not just SPARQL endpoints);
- Optional resource cache relying on the popular dogpile.cache library.

1.1.4 Status

OldMan is a young project **under active development** started in April 2014. Feel free to join us on Github and to subscribe to our mailing list *oldman AT librelist.com*.

Only Python 2.7 is currently supported, but support for Python 3.x is of course something we would like to consider.

1.1.5 Planned features

See our issue tracker.

Continue to *installation* or the *quickstart*.

1.2 Installation

Python 2.7 is required, so if in your distribution you have both Python 2.7 and Python 3.x, please make sure you are using the right version (usually, the command *python* links to Python 3.x).

1.2.1 Virtualenv

Because OldMan is still a young project, you will have to use development versions of some external libraries, such as RDFlib.

Thus, we recommend you to isolate the installation of OldMan and its dependencies by using Virtualenv.

If virtualenv is not already installed on your computer, you can install it with *easy_install* or *pip*:

```
$ sudo easy_install-2.7 install virtualenv
```

or:

```
$ sudo pip2 install virtualenv
```

Now create a directory where to install your virtualenv. For instance:

```
$ mkdir -p ~/envs/oldman-2.7
```

Move in, init and activae your virtualenv:

```
$ cd ~/envs/oldman-2.7
$ virtualenv2 .
$ source bin/activate
```

1.2.2 Install OldMan and its dependencies

```
$ mkdir src
$ cd src
$ git clone https://github.com/oldm/OldMan.git oldman
$ cd oldman
```

Install first the concrete requirements:

```
$ pip install -r requirements.txt
```

And then install oldman and its abstract (not yet fulfilled) dependencies:

```
$ python setup.py install
```

To test your installation, we encourage you to install the *nose* testing library:

```
$ pip install nose
```

You can run the tests:

```
$ nosetests tests/
```

Hope everything is ok!

Continue to the *quickstart* example.

1.3 Quickstart

1.3.1 Model creation

First, let's import some functions and classes:

```
from rdflib import Graph
from oldman import ResourceManager, parse_graph_safely, SPARQLDataStore
```

and create the RDF graph *schema_graph* that will contain our schema:

```
schema_graph = Graph()
```

The data graph is where we store the data that we generate. By default, it stores data in memory.

The role of the schema graph is to contain most of the domain logic necessary to build our models. In this example, we load it from a RDF file:

```
schema_url = "https://raw.githubusercontent.com/oldm/OldMan/master/examples/quickstart_schema.ttl"
parse_graph_safely(schema_graph, schema_url, format="turtle")
```

Another main piece of the domain logic is found in the JSON-LD context. Here, we just need its IRI:

```
context_iri = "https://raw.githubusercontent.com/oldm/OldMan/master/examples/quickstart_context.jsonl
```

We now have almost enough domain knowledge to create our models.

But first of all, we have to decide where to store our data. Here we create an in-memory RDF graph and use it as a SPARQL endpoint (`SPARQLDataStore`):

```
data_graph = Graph()
data_store = SPARQLDataStore(data_graph)
```

We extract the prefix information from the schema graph:

```
data_store.extract_prefixes(schema_graph)
```

Then we instantiate the central object of this framework, the `ResourceManager` object. Basically, it creates `Model` objects and offers convenient method to retrieve and create `Resource` objects:

```
manager = ResourceManager(schema_graph, data_store)
```

Finally, we create our *LocalPerson* `Model` object. For that, we need:

- The IRI or a JSON-LD term of the RDFS class of the model. Here *"LocalPerson"* is an alias for http://example.org/myvoc#LocalPerson defined in the context file ;
- The JSON-LD context;
- A prefix for creating the IRI of new resources (optional) ;
- An IRI fragment (optional);
- To declare that we want to generate incremental IRIs with short numbers for new `Resource` objects.

```
lp_model = manager.create_model("LocalPerson", context_iri,
                                iri_prefix="http://localhost/persons/",
                                iri_fragment="me", incremental_iri=True)
```

1.3.2 Resource editing

Now that the domain logic has been declared, we can create `Resource` objects for two persons, Alice and Bob:

```
alice = lp_model.create(name="Alice", emails={"alice@example.org"},
                        short_bio_en="I am ...")
bob = lp_model.new(name="Bob", blog="http://blog.example.com/",
                   short_bio_fr=u"J'ai grandi en ... .")
```

Alice is already stored in the *data_store* but not Bob. Actually, it cannot be saved yet because some information is still missing: its email addresses. This information is required by our domain logic. Let's satisfy this constraint and save Bob:

```
>>> bob.is_valid()
False
>>> bob.emails = {"bob@localhost", "bob@example.org"}
>>> bob.is_valid()
True
>>> bob.save()
```

Let's now declare that they are friends:

```
alice.friends = {bob}
bob.friends = {alice}
alice.save()
bob.save()
```

That's it. Have you seen many IRIs? Only one, for the blog. Let's look at them:

```
>>> alice.id
"http://localhost/persons/1#me"
>>> bob.id
"http://localhost/persons/2#me"
>>> bob.types
[u'http://example.org/myvoc#LocalPerson', u'http://xmlns.com/foaf/0.1/Person']
```

and at some other attributes:

```
>>> alice.name
"Alice"
>>> bob.emails
set(['bob@example.org', 'bob@localhost'])
>>> bob.short_bio_en
None
>>> bob.short_bio_fr
u"J'ai grandi en ... ."
```

We can assign an IRI when creating a `Resource` object:

```
>>> john_iri = "http://example.org/john#me"
>>> john = lp_model.create(id=john_iri, name="John", emails={"john@example.org"})
>>> john.id
"http://example.org/john#me"
```

1.3.3 Resource retrieval

By default, resource are not cached. We can retrieve Alice and Bob from the data graph as follows:

```
>>> alice_iri = alice.id
>>> # First person found named Bob
>>> bob = lp_model.get(name="Bob")
>>> alice = lp_model.get(id=alice_iri)

>>> # Or retrieve her as the unique friend of Bob
>>> alice = list(bob.friends)[0]
>>> alice.name
"Alice"
```

Finds all the persons:

```
>>> set(lp_model.all())
set([Resource(<http://example.org/john#me>), Resource(<http://localhost/persons/2#me>), Resource(<htt
>>> # Equivalent to
>>> set(lp_model.filter())
set([Resource(<http://localhost/persons/1#me>), Resource(<http://localhost/persons/2#me>), Resource(<
```

1.3.4 Serialization

JSON:

```
>>> print alice.to_json()
{
  "emails": [
    "alice@example.org"
  ],
  "friends": [
    "http://localhost/persons/2#me"
  ],
  "id": "http://localhost/persons/1#me",
  "name": "Alice",
  "short_bio_en": "I am ...",
  "types": [
    "http://example.org/myvoc#LocalPerson",
    "http://xmlns.com/foaf/0.1/Person"
  ]
}
```

JSON-LD:

```
>>> print john.to_jsonld()
{
  "@context": "https://raw.githubusercontent.com/oldm/OldMan/master/examples/quickstart_context.jsonl
  "emails": [
    "john@example.org"
  ],
  "id": "http://example.org/john#me",
  "name": "John",
  "types": [
    "http://example.org/myvoc#LocalPerson",
    "http://xmlns.com/foaf/0.1/Person"
  ]
}
```

Turtle:

```
>>> print bob.to_rdf("turtle")
@prefix bio: <http://purl.org/vocab/bio/0.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix myvoc: <http://example.org/myvoc#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost/persons/2#me> a myvoc:LocalPerson,
        foaf:Person ;
    bio:olb "J'ai grandi en ... ."@fr ;
```

```
    foaf:knows <http://localhost/persons/1#me> ;
    foaf:mbox "bob@example.org"^^xsd:string,
        "bob@localhost"^^xsd:string ;
    foaf:name "Bob"^^xsd:string ;
    foaf:weblog <http://blog.example.com/> .
```

1.3.5 Validation

Validation is also there:

```
>>> # Email is required
>>> lp_model.create(name="Jack")
oldman.exception.OMRequiredPropertyError: emails

>>> #Invalid email
>>> bob.emails = {'you_wont_email_me'}
oldman.exception.OMAttributeTypeCheckError: you_wont_email_me is not a valid email (bad format)

>>> # Not a set
>>> bob.emails = "bob@example.com"
oldman.exception.OMAttributeTypeCheckError: A container (<type 'set'>) was expected instead of <type

>>> #Invalid name
>>> bob.name = 5
oldman.exception.OMAttributeTypeCheckError: 5 is not a (<type 'str'>, <type 'unicode'>)
```

1.3.6 Domain logic

Here is the declared domain logic that we used:

JSON-LD context https://raw.githubusercontent.com/oldm/OldMan/master/examples/quickstart_context.jsonld:

```
{
  "@context": {
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "foaf": "http://xmlns.com/foaf/0.1/",
    "bio": "http://purl.org/vocab/bio/0.1/",
    "myvoc": "http://example.org/myvoc#",
    "Person": "foaf:Person",
    "LocalPerson": "myvoc:LocalPerson",
    "id": "@id",
    "types": "@type",
    "friends": {
      "@id": "foaf:knows",
      "@type": "@id",
      "@container": "@set"
    },
    "short_bio_fr": {
      "@id": "bio:olb",
      "@language": "fr"
    },
    "name": {
      "@id": "foaf:name",
      "@type": "xsd:string"
    },
    "emails": {
```

```
      "@id": "foaf:mbox",
      "@type": "xsd:string",
      "@container": "@set"
    },
    "blog": {
      "@id": "foaf:weblog",
      "@type": "@id"
    },
    "short_bio_en": {
      "@id": "bio:olb",
      "@language": "en"
    }
  }
}
```

Schema (uses the Hydra vocabulary) https://raw.githubusercontent.com/oldm/OldMan/master/examples/quickstart_schema.ttl:

```
@prefix bio: <http://purl.org/vocab/bio/0.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix hydra: <http://www.w3.org/ns/hydra/core#> .
@prefix myvoc: <http://example.org/myvoc#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# Properties that may be given to a foaf:Person (no requirement)
foaf:Person a hydra:Class ;
    hydra:supportedProperty [ hydra:property foaf:mbox ],
        [ hydra:property foaf:weblog ],
        [ hydra:property foaf:name ],
        [ hydra:property bio:olb ],
        [ hydra:property foaf:knows ].

# Local version of a Person with requirements
myvoc:LocalPerson a hydra:Class ;
    rdfs:subClassOf foaf:Person ;
    hydra:supportedProperty [ hydra:property foaf:mbox ;
            hydra:required true ],
        [ hydra:property foaf:name ;
            hydra:required true ].
```

1.4 Core concepts

1.4.1 Resource

A `Resource` object represents a Web resource identified by a regular IRI (internationalized URI) or or a skolem IRI (if it should treated as a blank node).

In OldMan, Web resources are described in conformance to the Resource Description Framework (RDF). A `Resource` object may have some attributes that provide the *predicate* (also called property) and the *object* terms of RDF triples describing the resource. The resource itself is the *subject* of the triple. Its attributes have arbitrary short names as defined in the JSON-LD context.

A `Resource` object access to its attributes through the `Model` objects to which it relates (through its `types`). Thus, if it has no *type* or its types that are not related to a `Model` object, a `Resource` object has no "RDF" attribute.

In OldMan, the relation between `Resource` and `Model` objects is *many-to-many*. It differs from traditional ORMs where the relation is *one-to-many* (the resource is usually an instance of the model and the latter is a Python class in

these frameworks). However, we expect that most `Resource` objects will relate to one `Model` object, but this is not a requirement. It is common for a resource in RDF to be instance of multiple RDFS classes so OldMan had to be ok with this practise.

Some inherited Python methods may also be provided by the `Model` objects.

Features

1. Edit its properties:

```
>>> # We assume that a model has been created for the RDFS class schema:Person.
>>> alice = Resource(resource_manager, types=["http://schema.org/Person"])
>>> alice.name = "Alice"
>>> print alice.name
Alice
>>> print alice.id
'http://localhost/person/3#me'
>>> alice.add_type("http://schema.org/Researcher")
>>> print alice.types
[u'http://schema.org/Person', u'http://schema.org/Researcher']
```

2. Persist its new values in the triplestore:

```
alice.save()
```

3. Call inherited methods:

```
alice.do_that()
```

4. Serialize to JSON, JSON-LD or any other RDF format:

```
>>> alice.to_jsonld()
{
  "@context": "https://example.com/context.jsonld",
  "id": "http://localhost/person/3#me",
  "name": "Alice",
  "types": [
    "http://schema.org/Person",
    "http://schema.org/Researcher"
  ]
}
>>> alice.to_rdf(format="turtle")
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost/persons/3#me> a schema:Person, schema:Researcher ;
            foaf:name "Alice"^^xsd:string .
```

1.4.2 ResourceManager

A `ResourceManager` object is the central object of OldMan.

It creates `Model` objects (`create_model()`) and retrieves `Resource` objects (`get()`, `filter()` and `sparql_filter()`).

It accepts Python method declarations if they happen before the creation of `Model` objects (`declare_method()`).

It also provide helper functions to create new `Resource` objects (`create()` and `new()`) but it is usually simpler to use those of a `Model` object.

For creating the `ResourceManager` object, the schema graph and the data store (`DataStore`) must be given.

Basically, the schema graph describes which properties should be expected for a given RDFS class, which are required and what are the constraints.

1.4.3 Model

In OldMan, models are not Python classes but `Model` objects. However, on the RDF side, they correspond to RDFS classes (their `class_iri` attributes).

Their main role is to provide attributes and methods to `Resource` objects, as explained above.

`Model` objects are created by the `ResourceManager` object.

A model provide some helpers above the `ResourceManager` object (`get()`, `filter()`, `new()` and `create()`) that include the `class_iri` to the *types* parameter of these methods.

1.4.4 DataStore

A `DataStore` implements the CRUD operations on Web Resources exposed by the `ResourceManager` and `Model` objects.

The vision of OldMan is to include a large choice of data stores. But currently, only SPARQL endpoints are supported.

Non-CRUD operations may also be introduced in the future (in discussion).

Any data store accepts a `dogpile.cache.region.CacheRegion` object to enable its `ResourceCache` object. By default the latter is disabled so it does not cache the `Resource` objects loaded from and stored in the data store.

SPARQLDataStore

A `SPARQLDataStore` object relies on one or two RDF graphs (`rdflib.graph.Graph`): the data and default graphs.

The data graph is where regular resources are saved and loaded.

The default graph (`rdflib.graph.ConjunctiveGraph` or `rdflib.graph.Dataset`) may be given as an optional second graph. Its only constraint is to include the content of the data graph in its default graph.

1.5 Examples

1.5.1 DBpedia querying (read-only)

Source code

This example presents a use case where an OLDM produces a significant overhead that is important to understand.

We want to query the DBpedia which contains RDF statements extracted from the info-boxes of Wikipedia. DBpedia provides a public SPARQL endpoint powered by Virtuoso.

Inspired by a gist of O. Berger, we will display:

1. The 10 first French films found on DBpedia and the names of their actors;
2. The films in which Michel Piccoli had a role.

Direct SPARQL queries (without OldMan)

First, let's create a Graph to access the DBpedia SPARQL endpoint

```
from rdflib import Graph
from rdflib.plugins.stores.sparqlstore import SPARQLStore
data_graph = Graph(SPARQLStore("http://dbpedia.org/sparql", context_aware=False))
```

Query 1

```
import time
q3_start_time = time.time()

results = data_graph.query("""
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpo: <http://dbpedia.org/ontology/>

SELECT ?film ?title_fr ?title_en ?actor ?actor_name_fr ?actor_name_en
WHERE {
    {
     SELECT ?film
     WHERE {
        ?film a dbpo:Film ;
              dcterms:subject <http://dbpedia.org/resource/Category:French_films> .
      }
      LIMIT 10
    }
    OPTIONAL {
       ?film rdfs:label ?title_en .
       FILTER langMatches( lang(?title_en), "EN" ) .
    }
    OPTIONAL {
       ?film rdfs:label ?title_fr .
       FILTER langMatches( lang(?title_fr), "FR" ) .
    }
    OPTIONAL {
      ?film dbpo:with ?actor .
      OPTIONAL {
        ?actor foaf:name ?actor_name_en .
        FILTER langMatches( lang(?actor_name_en), "EN" ) .
      }
      OPTIONAL {
        ?actor foaf:name ?actor_name_fr .
        FILTER langMatches( lang(?actor_name_fr), "FR" ) .
      }
    }
}
""")
```

Now we extract the film titles and the names of the actors:

```
film_titles = {}
film_actors = {}
for film_iri, title_fr, title_en, actor_iri, actor_name_fr, actor_name_en in results:
    if film_iri not in film_titles:
```

```
        for t in [title_fr, title_en, film_iri]:
            if t is not None:
                film_titles[film_iri] = unicode(t)
                break
    for name in [actor_name_fr, actor_name_en, actor_iri]:
        if name is not None:
            if film_iri not in film_actors:
                film_actors[film_iri] = [name]
            elif name not in film_actors[film_iri]:
                film_actors[film_iri].append(unicode(name))
            break
```

and display them:

```
>>> for film_iri in film_titles:
...     title = film_titles[film_iri]
...     if film_iri not in film_actors:
...         print "%s %s (no actor declared)" % (title, film_iri)
...     else:
...         actor_names = ", ".join(film_actors[film_iri])
...         print "%s with %s" % (title, actor_names)
And Now... Ladies and Gentlemen with Patricia Kaas, Jeremy Irons, Thierry Lhermitte
Un long dimanche de fiançailles (film) with Dominique Pinon, Marion Cotillard, Ticky Holgado, Audrey
Charlotte et Véronique http://dbpedia.org/resource/All_the_Boys_Are_Called_Patrick (no actor declared
Toutes ces belles promesses with Jeanne Balibar, Bulle Ogier, Valerie Crunchant, http://dbpedia.org/r
Édith et Marcel with Évelyne Bouix, Evelyne Bouix, http://dbpedia.org/resource/Marcel_Cerdan_Jr
Une robe d'été http://dbpedia.org/resource/A_Summer_Dress (no actor declared)
9 semaines 1/2 with Kim Basinger, Mickey Rourke
Tout sur ma mère with Penélope Cruz, Penélope Cruz Sánchez, Cecilia Roth, Antonia San Juan, Candela P
Artemisia (film) with Miki Manojlović, Predrag Miki Manojlovic, Michel Serrault, Valentina Cervi
Two Days in Paris with Julie Delpy, Adam Goldberg, Daniel Bruhl
>>> print "Done in %.3f seconds" % (time.time() - q3_start_time)
Done in 0.252 seconds
```

Some names are missing in the DBpedia and are replaced by the URI. The film URI is also displayed when the actors are unknown so that you can check with your browser that this information is missing.

Query 2

```
q4_start_time = time.time()
results = data_graph.query("""
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpo: <http://dbpedia.org/ontology/>

SELECT ?film ?title_fr ?title_en
WHERE {
    ?film a dbpo:Film ;
          dbpo:with <http://dbpedia.org/resource/Michel_Piccoli> .
    OPTIONAL {
       ?film rdfs:label ?title_en .
       FILTER langMatches( lang(?title_en), "EN" ) .
    }
    OPTIONAL {
       ?film rdfs:label ?title_fr .
       FILTER langMatches( lang(?title_fr), "FR" ) .
```

```
    }
}
""")

>>> for film_iri, title_fr, title_en in results:
...     if film_iri not in film_titles:
...         for t in [title_fr, title_en, film_iri]:
...             if t is not None:
...                 print t
...                 break
    La Diagonale du fou
    Le Journal d'une femme de chambre (film, 1964)
    La Grande Bouffe
    Max et les Ferrailleurs
    La Voie lactée (film, 1969)
    Les Demoiselles de Rochefort
    Le Saut dans le vide
    Belle toujours
    Boxes
    Des enfants gâtés
    Une étrange affaire
    Belle de Jour (film)
    Benjamin ou les Mémoires d'un puceau
    Le Mépris (film)
    Dillinger est mort
    Généalogies d'un crime
    Je rentre à la maison
    La Belle Noiseuse
    La Chamade (film)
    Le Prix du danger (film)
    Mauvais Sang (film)
    Milou en mai
    Passion (film, 1982)
    La Prophétie des grenouilles
    La Poussière du temps
    Le Fantôme de la liberté
    Compartiment tueurs
    Les Choses de la vie
    Themroc
    Une chambre en ville
    Vincent, François, Paul... et les autres
    Habemus papam (film)
    Les Noces rouges
    Les Cent et Une Nuits de Simon Cinéma
    La Décade prodigieuse
    Der Preis fürs Überleben
    Party (1996 film)
    The Distant Land
    Passion in the Desert
>>> print "Done in %.3f seconds" % (time.time() - q4_start_time)
Done in 0.180 seconds
```

With OldMan

Let's first create two `Model` objects: *film_model* and *person_model* from these context and schema:

```
from oldman import ResourceManager, SPARQLDataStore
from dogpile.cache import make_region

schema_url = "https://raw.githubusercontent.com/oldm/OldMan/master/examples/dbpedia_film_schema.ttl"
schema_graph = Graph().parse(schema_url, format="turtle")

context_url = "https://raw.githubusercontent.com/oldm/OldMan/master/examples/dbpedia_film_context.jso

# In-memory cache
cache_region = make_region().configure('dogpile.cache.memory_pickle')

# SPARQL data store
data_store = SPARQLDataStore(data_graph, cache_region=cache_region)

# Resource Manager and Models
manager = ResourceManager(schema_graph, data_store)
film_model = manager.create_model("http://dbpedia.org/ontology/Film", context_url)
# JSON-LD terms can be used instead of IRIs
person_model = manager.create_model("Person", context_url)
```

Please note that we set up a resource cache and reused the *data_graph*.

We also declare two extraction functions:

```
def extract_title(film):
    if len(film.titles) > 0:
        key = "fr" if "fr" in film.titles else film.titles.keys()[0]
        return "%s (%s version)" % (film.titles[key], key)
    return film.id

def extract_name(person):
    if person.names is not None and len(person.names) > 0:
        for key in ["fr", "en"]:
            if key in person.names:
                return person.names[key]
        return person.names.values()[0]
    return person.id
```

Query 1 (lazy)

By default, OldMan behaves lazily:

```
>>> q1_start_time = time.time()
>>> for film in film_model.filter(subjects=["http://dbpedia.org/resource/Category:French_films"],
...                               limit=10):
...     title = extract_title(film)
...     if film.actors is None:
...         print "   %s %s (no actor declared)" % (title, film.id)
...     else:
...         actor_names = ", ".join([extract_name(a) for a in film.actors])
...         print "%s with %s" % (title, actor_names)
Édith et Marcel (fr version) with http://dbpedia.org/resource/Marcel_Cerdan_Jr, Evelyne Bouix
Two Days in Paris (fr version) with Julie Delpy, Adam Goldberg, Daniel Bruhl
9 semaines 1/2 (fr version) with Kim Basinger, Mickey Rourke
Une robe d'été (fr version) http://dbpedia.org/resource/A_Summer_Dress (no actor declared)
Un long dimanche de fiançailles (film) (fr version) with Jodie Foster, Chantal Neuwirth, Marion Cotil
Tout sur ma mère (fr version) with Cecilia Roth, Antonia San Juan, Marisa Paredes, Candela Pena, Pene
```

```
Charlotte et Véronique (fr version) http://dbpedia.org/resource/All_the_Boys_Are_Called_Patrick (no a
Toutes ces belles promesses (fr version) with Valerie Crunchant, Jeanne Balibar, Bulle Ogier, http:/
And Now... Ladies and Gentlemen (fr version) with Thierry Lhermitte, Jeremy Irons, Patricia Kaas
Artemisia (film) (fr version) with Michel Serrault, Miki Manojlović, Valentina Cervi
>>> print "Done in %.3f seconds" % (time.time() - q1_start_time)
Done in 17.123 seconds
```

17s? Why is it so slow? There are two reasons:

1. OldMan loads a `Resource` object for each film or actor that is displayed. Loading a `Resource` object implies to retrieve all the triples in which the resource is the subject. In DBpedia, entries like films and actors have often many triples. Some of them have long textual literal values (localized paragraphs from Wikipedia). For instance, see http://dbpedia.org/resource/Penelope_Cruz. This approach retrieves much more information than we need for our specific query.
2. By default OldMan is lazy so it retrieves each a `Resource` object at the last time, *one by one in sequence*. The execution of this long sequence of queries takes a long time, partly because of the network latency that is multiplied by the number of queries.

Query 1 (eager)

While this first phenomenon is something you should expect when using an OLDM, the second reason can avoided by adopting an eager strategy:

```
>>> q1_start_time = time.time()
>>> for film in film_model.filter(subjects=["http://dbpedia.org/resource/Category:French_films"],
...                               limit=10, eager=True,
...                               pre_cache_properties=["http://dbpedia.org/ontology/starring"]):
... # Code and results not shown
>>> print "Done in %.3f seconds" % (time.time() - q1_start_time)
Done in 2.518 seconds
```

The eager strategy makes one heavy SPARQL request that returns all the triples about the films but also about the actors (thanks to the pre-cached property *dbpo:starring*). The network latency is then almost minimal.

If we re-query it again lazily, thanks to the cache it makes just one lightweight SPARQL query:

```
>>> q1_start_time = time.time()
>>> for film in film_model.filter(subjects=["http://dbpedia.org/resource/Category:French_films"],
...                               limit=10):
... # Code and results not shown
>>> print "Done in %.3f seconds" % (time.time() - q1_start_time)
Done in 0.182 seconds
```

But if we re-query it eagerly, the heavy query will be sent again. The cache is then of little interest:

```
>>> # Code and results not shown
>>> print "Done in %.3f seconds" % (time.time() - q1_start_time)
Done in 2.169 seconds
```

Query 2 (lazy)

```
>>> q2_start_time = time.time()
>>> for film in film_model.filter(actors=["http://dbpedia.org/resource/Michel_Piccoli"]):
...     print extract_title(film)
... # Results not shown
```

```
>>> print "Done in %.3f seconds" % (time.time() - q2_start_time)
Done in 16.419 seconds
```

Query 2 (eager)

```
>>> q2_start_time = time.time()
>>> for film in film_model.filter(actors=["http://dbpedia.org/resource/Michel_Piccoli"],
                                  eager=True):
... # Code and results not shown
>>> print "Done in %.3f seconds" % (time.time() - q2_start_time)
Done in 1.503 seconds
```

CHAPTER 2

API reference

Main classes manipulated by end-users: `ResourceManager`, `Model` and `Resource`.

`IriGenerator` classes can be found in the `oldman.iri` module.

`DataStore` classes can be found in the package `oldman.store.datastore`.

2.1 oldman package

2.1.1 oldman.attribute module

class `oldman.attribute.`**`OMAttribute`**(*manager*, *metadata*, *value_format*)

Bases: `object`

An `OMAttribute` object corresponds to a JSON-LD term that refers to a RDF property.

Technically, the name of the `OMAttribute` object is a JSON-LD term, namely *"a short-hand string that expands to an IRI or a blank node identifier"* (cf. the JSON-LD standard) which corresponds here to a RDF property (see `OMProperty`).

In JSON-LD, the same RDF property may correspond to multiple JSON-LD terms that have different metadata. For instance, a foaf:Person resource may have two attributes for its bio in English and in French. These attributes have two different languages but use the same property *bio:olb*. Look at the quickstart example to see it in practice.

An `OMAttribute` object manages the values of every `Resource` object that depends on a given `Model` object.

Each value may be :

•*None*;

•The Python equivalent for a RDF literal (double, string, date, etc.);

•An IRI;

•A collection (set, list and dict) of these types.

Parameters

- **manager** – `ResourceManager` object.
- **metadata** – `OMAttributeMetadata` object.

- **value_format** – `ValueFormat` object that validates the format of values and converts RDF values into regular Python objects.

check_validity(*resource*, *is_end_user=True*)
Raises an `OMEditError` exception if the attribute value assigned to a resource is invalid.

Parameters

- **resource** – `Resource` object.
- **is_end_user** – *False* when an authorized user (not a regular end-user) wants to force some rights. Defaults to *True*.

check_value(*value*)
Checks a new **when assigned**.

Raises an `oldman.exception.OMAttributeTypeCheckError` exception if the value is invalid.

Parameters **value** – collection or atomic value.

container
JSON-LD container (*"@set"*, *"@list"*, *"@language"* or *"@index"*). May be *None*.

delete_former_value(*resource*)
Clears the former value that has been replaced.

Parameters **resource** – `Resource` object.

get(*resource*)
Gets the attribute value of a resource.

Parameters **resource** – `Resource` object.

Returns Atomic value or a generator.

get_former_value(*resource*)
Gets out the former value that has been replaced.

Parameters **resource** – `Resource` object.

Returns its former attribute value or *None*.

get_lightly(*resource*)
Gets the attribute value of a resource in a lightweight manner.

By default, behaves exactly like `get()`. See the latter function for further details.

has_new_value(*resource*)

Parameters **resource** – `Resource` object.

has_value(*resource*)
Tests if the resource attribute has a non-None value.

Parameters **resource** – `Resource` object.

Returns *False* if the value is *None*.

is_read_only
True if the property cannot be modified by regular end-users.

is_required
True if its property is required.

is_valid(*resource*, *is_end_user=True*)
Tests if the attribute value assigned to a resource is valid.

See `check_validity()` for further details.

Returns *False* if the value assigned to the resource is invalid and *True* otherwise.

is_write_only
True if the property cannot be accessed by regular end-users.

jsonld_type
JSON-LD type (datatype IRI or JSON-LD keyword). May be *None*.

language
Its language if localized.

manager
Its `ResourceManager` object.

name
Its name as an attribute.

om_property
`OMProperty` to which it belongs.

other_attributes
Other `OMAttribute` objects of the same property.

reversed
True if the object and subject in RDF triples should be reversed.

set(*resource*, *value*)
Sets the attribute value of a resource.

Parameters

- **resource** – `Resource` object.
- **value** – Its value for this attribute.

to_nt(*resource*)
Converts its current attribute value to N-Triples (NT) triples.

Relies on `value_to_nt()`.

Parameters **resource** – `Resource` object.

Returns N-Triples serialization of its attribute value.

update_from_graph(*resource*, *sub_graph*, *initial=False*)
Updates a resource attribute value by extracting the relevant information from a RDF graph.

Parameters

- **resource** – `Resource` object.
- **sub_graph** – `rdflib.Graph` object containing the value to extract.
- **initial** – *True* when the value is directly from the datastore. Defaults to *False*.

value_format
`ValueFormat` object that validates the format of values and converts RDF values into regular Python objects.

value_to_nt(*value*)
Converts value(s) to N-Triples (NT) triples.

Parameters **value** – Value of property.

Returns N-Triples serialization of this value.

class oldman.attribute.**OMAttributeMetadata**
Bases: tuple

OMAttributeMetadata(name, property, language, jsonld_type, container, reversed)

container
Alias for field number 4

jsonld_type
Alias for field number 3

language
Alias for field number 2

name
Alias for field number 0

property
Alias for field number 1

reversed
Alias for field number 5

class oldman.attribute.**ObjectOMAttribute**(*manager*, *metadata*, *value_format*)
Bases: oldman.attribute.OMAttribute

An ObjectOMAttribute object is an OMAttribute object that depends on an owl:ObjectProperty.

get(*resource*)
See get().

Returns Resource object or a generator of Resource objects.

get_lightly(*resource*)
Gets the attribute value of a resource in a lightweight manner.

By contrast with get() only IRIs are returned, not Resource objects.

Returns An IRI, a list or a set of IRIs or *None*.

set(*resource*, *value*)
See set().

Accepts Resource object(s) or IRI(s).

2.1.2 oldman.exception module

exception oldman.exception.**AlreadyAllocatedModelError**
Bases: oldman.exception.ModelGenerationError

The class IRI or the short name of a new model is already allocated.

exception oldman.exception.**ModelGenerationError**
Bases: oldman.exception.OMError

Error occured when generating a new model.

exception oldman.exception.**OMAccessError**
Bases: oldman.exception.OMUserError

Error when accessing objects.

exception oldman.exception.**OMAlreadyDeclaredDatatypeError**
Bases: oldman.exception.OMAttributeDefError

At least two different datatypes for the same attribute.

You may check the possible datatype inherited from the property (rdfs:range) and the one specified in the JSON-LD context.

exception oldman.exception.**OMAlreadyGeneratedAttributeError**
Bases: oldman.exception.OMInternalError

Attribute generation occurs only once per SupportedProperty.

You should not try to add metadata or regenerate after that.

exception oldman.exception.**OMAttributeAccessError**
Bases: oldman.exception.OMAccessError

When such an attribute cannot be identified (is not supported or no model has been found).

exception oldman.exception.**OMAttributeDefError**
Bases: oldman.exception.OMSchemaError

Inconsistency in the definition of a model class attribute.

exception oldman.exception.**OMAttributeTypeCheckError**
Bases: oldman.exception.OMEditError

The value assigned to the attribute has wrong type.

exception oldman.exception.**OMClassInstanceError**
Bases: oldman.exception.OMAccessError

The object is not an instance of the expected RDFS class.

exception oldman.exception.**OMDataStoreError**
Bases: oldman.exception.OMError

Error detected in the stored data.

exception oldman.exception.**OMDifferentHashlessIRIError**
Bases: oldman.exception.OMEditError

When creating or updating an object with a different hashless IRI is forbidden.

Blank nodes are not concerned.

exception oldman.exception.**OMEditError**
Bases: oldman.exception.OMUserError

Runtime errors, occuring when editing or creating an object.

exception oldman.exception.**OMError**
Bases: exceptions.Exception

Root of exceptions generated by the oldman package.

exception oldman.exception.**OMExpiredMethodDeclarationTimeSlotError**
Bases: oldman.exception.ModelGenerationError

All methods must be declared before creating a first model.

exception oldman.exception.**OMForbiddenSkolemizedIRIError**
Bases: oldman.exception.OMEditError

When updating a skolemized IRI from the local domain is forbidden.

exception oldman.exception.**OMHashIriError**
Bases: oldman.exception.OMAccessError

A hash IRI has been given instead of a hash-less IRI.

exception oldman.exception.**OMInternalError**
Bases: oldman.exception.OMError

Do not expect it.

exception oldman.exception.**OMObjectNotFoundError**
Bases: oldman.exception.OMAccessError

When the object is not found.

exception oldman.exception.**OMPropertyDefError**
Bases: oldman.exception.OMSchemaError

Inconsistency in the definition of a supported property.

exception oldman.exception.**OMPropertyDefTypeError**
Bases: oldman.exception.OMPropertyDefError

A RDF property cannot be both an ObjectProperty and a DatatypeProperty.

exception oldman.exception.**OMReadOnlyAttributeError**
Bases: oldman.exception.OMEditError

End-users are not allowed to edit this attribute.

exception oldman.exception.**OMRequiredHashlessIRIError**
Bases: oldman.exception.OMEditError

No hash-less IRI has been given.

exception oldman.exception.**OMRequiredPropertyError**
Bases: oldman.exception.OMEditError

A required property has no value.

exception oldman.exception.**OMReservedAttributeNameError**
Bases: oldman.exception.OMAttributeDefError

Some attribute names are reserved and should not be included in the JSON-LD context.

exception oldman.exception.**OMSPARQLError**
Bases: oldman.exception.OMAccessError

Invalid SPARQL query given.

exception oldman.exception.**OMSPARQLParseError**
Bases: oldman.exception.OMInternalError

Invalid SPARQL request.

exception oldman.exception.**OMSchemaError**
Bases: oldman.exception.ModelGenerationError

Error in the schema graph and/or the JSON-LD context.

exception oldman.exception.**OMUnauthorizedTypeChangeError**
Bases: oldman.exception.OMEditError

When updating a resource with new types without explicit authorization.

exception oldman.exception.**OMUndeclaredClassNameError**
Bases: oldman.exception.ModelGenerationError

The name of the model class should be defined in the JSON-LD context.

exception oldman.exception.**OMUniquenessError**
Bases: oldman.exception.OMEditError

Attribute uniqueness violation.

Example: IRI illegal reusing.

exception oldman.exception.**OMUserError**
Bases: oldman.exception.OMError

Error when accessing or editing objects.

exception oldman.exception.**OMWrongResourceError**
Bases: oldman.exception.OMEditError

Not updating the right object.

exception oldman.exception.**UnsupportedDataStorageFeatureException**
Bases: oldman.exception.OMDataStoreError

Feature not supported by the data store.

2.1.3 oldman.iri module

class oldman.iri.**BlankNodeIriGenerator**(*hostname=u'localhost'*)
Bases: oldman.iri.PrefixedUUIDIriGenerator

Generates skolem IRIs that denote blank nodes.

Parameters **hostname** – Defaults to *"localhost"*.

class oldman.iri.**IncrementalIriGenerator**(*prefix*, *data_store*, *class_iri*, *fragment=None*)
Bases: oldman.iri.IriGenerator

Generates IRIs with short numbers.

Beautiful but **slow** in concurrent settings. The number generation implies a critical section and a sequence of two SPARQL requests, which represents a significant bottleneck.

Parameters

- **prefix** – IRI prefix.
- **graph** – rdflib.Graph object where to store the counter.
- **class_iri** – IRI of the RDFS class of which new Resource objects are instance of. Usually corresponds to the class IRI of the Model object that owns this generator.
- **fragment** – IRI fragment to append to the hash-less IRI. Defaults to *None*.

generate(***kwargs*)
See oldman.iri.IriGenerator.generate().

reset_counter()
For test purposes only

class oldman.iri.**IriGenerator**
Bases: object

An IriGenerator object generates the IRIs of some new Resource objects.

generate(***kwargs*)
Generates an IRI.

Returns Unique IRI (unicode string).

class oldman.iri.**PrefixedUUIDIriGenerator**(*prefix*, *fragment=None*)
Bases: oldman.iri.IriGenerator

Uses a prefix, a fragment and a unique UUID1 number to generate IRIs.

Recommended generator because UUID1 is robust and fast (no DB access).

Parameters

- **prefix** – IRI prefix.
- **fragment** – IRI fragment to append to the hash-less IRI. Defaults to *None*.

generate(***kwargs*)
See oldman.iri.IriGenerator.generate().

class oldman.iri.**UUIDFragmentIriGenerator**
Bases: oldman.iri.IriGenerator

Generates an hashed IRI from a hash-less IRI.

Its fragment is a unique UUID1 number.

generate(*hashless_iri*)
See oldman.iri.IriGenerator.generate().

2.1.4 oldman.model module

class oldman.model.**Model**(*manager*, *name*, *class_iri*, *ancestry_iris*, *context*, *om_attributes*, *id_generator*, *methods=None*)
Bases: object

A Model object represents a RDFS class on the Python side.

It gathers OMAttribute objects and Python methods which are made available to Resource objects that are instances of its RDFS class.

It also creates and retrieves Resource objects that are instances of its RDFS class. It manages an IriGenerator object.

Model creation

Model objects are normally created by a ResourceManager object. Please use the oldman.management.manager.ResourceManager.create_model() method for creating new Model objects.

Parameters

- **manager** – ResourceManager object that has created this model.
- **name** – Model name. Usually corresponds to a JSON-LD term or to a class IRI.
- **class_iri** – IRI of the RDFS class represented by this Model object.
- **ancestry_iris** – ancestry of the attribute *class_iri*. Each instance of *class_iri* is also instance of these classes.

- **context** – An IRI, a *list* or a *dict* that describes the JSON-LD context. See http://www.w3.org/TR/json-ld/#the-context for more details.
- **om_attributes** – *dict* of `OMAttribute` objects. Keys are their names.
- **id_generator** – `IriGenerator` object that generates IRIs from new `Resource` objects.
- **methods** – *dict* of Python functions that takes as first argument a `Resource` object. Keys are the method names. Defaults to *{}*.

access_attribute(*name*)
Gets an `OMAttribute` object.

Used by the `Resource` class but an end-user should not need to call it.

Parameters **name** – Name of the attribute.

Returns The corresponding `OMAttribute` object.

all(*limit=None*, *eager=False*)
Finds every `Resource` object that is instance of its RDFS class.

Parameters

- **limit** – Upper bound on the number of solutions returned (SPARQL LIMIT). Positive integer. Defaults to *None*.
- **eager** – If *True* loads all the Resource objects within one single SPARQL query. Defaults to *False* (lazy).

Returns A generator of `Resource` objects.

ancestry_iris
IRIs of the ancestry of the attribute *class_iri*.

class_iri
IRI of the class IRI the model refers to.

context
An IRI, a *list* or a *dict* that describes the JSON-LD context. See http://www.w3.org/TR/json-ld/#the-context for more details.

create(*id=None*, *hashless_iri=None*, ***kwargs*)
Creates a new resource and saves it.

See `new()` for more details.

filter(*hashless_iri=None*, *limit=None*, *eager=False*, *pre_cache_properties=None*, ***kwargs*)
Finds the `Resource` objects matching the given criteria.

The *class_iri* attribute is added to the *types*.

See `oldman.management.finder.ResourceFinder.filter()` for further details.

generate_iri(***kwargs*)
Generates a new IRI.

Used by the `Resource` class but an end-user should not need to call it.

Returns A new IRI.

get(*id=None*, *hashless_iri=None*, ***kwargs*)
Gets the first `Resource` object matching the given criteria.

The *class_iri* attribute is added to the *types*. Also looks if reversed attributes should be considered eagerly.

See `oldman.management.finder.Finder.get()` for further details.

has_reversed_attributes
Is *True* if one of its attributes is reversed.

is_subclass_of(*model*)
Returns *True* if its RDFS class is a sub-class *(rdfs:subClassOf)* of the RDFS class of another model.

Parameters **model** – `Model` object to compare with.

Returns *True* if is a sub-class of the other model, *False* otherwise.

methods
dict of Python functions that takes as first argument a `Resource` object. Keys are the method names.

name
Name attribute.

new(*id=None*, *hashless_iri=None*, ***kwargs*)
Creates a new `Resource` object without saving it.

The *class_iri* attribute is added to the *types*.

See `new()` for more details.

om_attributes
dict of `OMAttribute` objects. Keys are their names.

reset_counter()
Resets the counter of the IRI generator.

Please use it only for test purposes.

`oldman.model.`**clean_context**(*context*)
Cleans the context.

Context can be an IRI, a *list* or a *dict*.

2.1.5 oldman.property module

class `oldman.property.`**OMProperty**(*manager*, *property_iri*, *supporter_class_iri*, *is_required=False*, *read_only=False*, *write_only=False*, *reversed=False*, *cardinality=None*, *property_type=None*, *domains=None*, *ranges=None*)
Bases: `object`

An `OMProperty` object represents the support of a RDF property by a RDFS class.

It gathers some `OMAttribute` objects (usually one).

An `OMProperty` object is in charge of generating its `OMAttribute` objects according to the metadata that has been extracted from the schema and JSON-LD context.

A property can be reversed: the `Resource` object to which the `OMAttribute` objects will be (indirectly) attached is then the object of this property, not its subject (?o ?p ?s).

Consequently, two `OMProperty` objects can refer to the same RDF property when one is reversed while the second is not.

Parameters

- **manager** – `ResourceManager` object.
- **property_iri** – IRI of the RDF property.
- **supporter_class_iri** – IRI of the RDFS class that supports the property.

- **is_required** – If *True* instances of the supporter class must assign a value to this property for being valid. Defaults to *False*.
- **read_only** – If *True*, the value of the property cannot be modified by a regular end-user. Defaults to *False*.
- **write_only** – If *True*, the value of the property cannot be read by a regular end-user. Defaults to *False*.
- **reversed** – If *True*, the property is reversed. Defaults to *False*.
- **cardinality** – Defaults to *None*. Not yet supported.
- **property_type** – String. In OWL, a property is either a DatatypeProperty or an ObjectProperty. Defaults to *None* (unknown).
- **domains** – Set of class IRIs that are declared as the RDFS domain of the property. Defaults to *set()*.
- **ranges** – Set of class IRIs that are declared as the RDFS range of the property. Defaults to *set()*.

add_attribute_metadata(*name*, *jsonld_type=None*, *language=None*, *container=None*, *reversed=False*)

Adds metadata about a future `OMAttribute` object.

Parameters

- **name** – JSON-LD term representing the attribute.
- **jsonld_type** – JSON-LD type (datatype IRI or JSON-LD keyword). Defaults to *None*.
- **language** – Defaults to *None*.
- **container** – JSON-LD container (*"@set"*, *"@list"*, *"@language"* or *"@index"*). Defaults to *None*.
- **reversed** – *True* if the object and subject in RDF triples should be reversed. Defaults to *False*.

add_domain(*domain*)

Declares a RDFS class as part of the domain of the property.

Parameters **domain** – IRI of RDFS class.

add_range(*p_range*)

Declares a RDFS class as part of the range of the property.

Parameters **p_range** – IRI of RDFS class.

declare_is_required()

Makes the property be required. Is irreversible.

default_datatype

IRI that is the default datatype of the property.

May be *None* (if not defined or if the property is an owl:ObjectProperty)

domains

Set of class IRIs that are declared as the RDFS domain of the property.

generate_attributes(*attr_format_selector*)

Generates its `OMAttribute` objects.

Can be called only once. When called a second time, raises an `OMAlreadyGeneratedAttributeError` exception.

Parameters **attr_format_selector** – `ValueFormatSelector` object.

iri
IRI of RDF property.

is_read_only
True if the property cannot be modified by regular end-users.

is_required
True if the property is required.

is_write_only
True if the property cannot be accessed by regular end-users.

om_attributes
Set of `OMAttribute` objects that depends on this property.

ranges
Set of class IRIs that are declared as the RDFS range of the property.

reversed
True if the property is reversed (?o ?p ?s).

supporter_class_iri
IRI of the RDFS class that supports the property.

type
The property can be a owl:DatatypeProperty (*"datatype"*) or an owl:ObjectProperty (*"object"*). Sometimes its type is unknown (*None*).

2.1.6 oldman.resource module

class `oldman.resource.`**Resource**(*manager*, *id=None*, *types=None*, *hashless_iri=None*, *is_new=True*, ***kwargs*)

Bases: `object`

A `Resource` object is a subject-centric representation of a Web resource. A set of `Resource` objects is equivalent to a RDF graph.

In RDF, a resource is identified by an IRI (globally) or a blank node (locally). Because blank node support is complex and limited (`rdflib.plugins.stores.sparqlstore.SPARQLStore` stores do not support them), **every** `Resource` **object has an IRI**.

This IRI is either given or generated by a `IriGenerator` object. Some generators generate recognizable skolem IRIs that are treated as blank nodes when the resource is serialized into JSON, JSON-LD or another RDF format (for external consumption).

A resource is usually instance of some RDFS classes. These classes are grouped in its attribute *types*. `Model` objects are found from these classes, by calling the method `oldman.management.manager.ResourceManager.find_models_and_types()`. Models give access to Python methods and to `OMAttribute` objects. Their ordering determines inheritance priorities. The main model is the first one of this list.

Values of `OMAttribute` objects are accessible and modifiable like ordinary Python attribute values. However, these values are checked so some `OMAccessError` or `OMEditError` errors may be raised.

Example:

```
>>> alice = Resource(manager, types=["http://schema.org/Person"], name=u"Alice")
>>> alice.id
u'http://localhost/persons/1'
```

```
>>> alice.name
u'Alice'
>>> alice.save()
>>> alice.name = "Alice A."
>>> print alice.to_jsonld()
{
   "@context": "http://localhost/person.jsonld",
   "id": "http://localhost/persons/1",
   "types": [
              "http://schema.org/Person"
            ],
   "name": "Alice A."
}
>>> alice.name = 5
oldman.exception.OMAttributeTypeCheckError: 5 is not a (<type 'str'>, <type 'unicode'>)
```

Resource creation

`Resource` objects are normally created by a `Model` or a `ResourceManager` object. Please use the methods `oldman.model.Model.create()`, `oldman.model.Model.new()`, `oldman.management.manager.ResourceManager.create()` or `oldman.management.manager.ResourceManager.new()` for creating new `Resource` objects.

Parameters

- **manager** – `ResourceManager` object. Gives access to the *data_graph* (where the triples are stored), the *union_graph* and the *resource_cache*.
- **id** – IRI of the resource. If not given, this IRI is generated by the main model. Defaults to *None*.
- **types** – IRI list or set of the RDFS classes the resource is instance of. Defaults to *set()*.
- **hashless_iri** – Hash-less IRI that is given to the main model for generating a new IRI if no *id* is given. The IRI generator may ignore it. Defaults to *None*.
- **is_new** – When is *True* and *id* given, checks that the IRI is not already existing in the *union_graph*. Defaults to *True*.
- **kwargs** – values indexed by their attribute names.

add_type(*additional_type*)

Declares that the resource is instance of another RDFS class.

Note that it may introduce a new model to the list and change its ordering.

Parameters **additional_type** – IRI or JSON-LD term identifying a RDFS class.

check_validity()

Checks its validity.

Raises an `oldman.exception.OMEditError` exception if invalid.

context

An IRI, a *list* or a *dict* that describes the JSON-LD context.

Derived from `oldman.model.Model.context` attributes.

delete()

Removes the resource from the *data_graph* and the *resource_cache*.

Cascade deletion is done for related resources satisfying the test `should_delete_resource()`.

full_update(*full_dict*, *is_end_user=True*, *allow_new_type=False*, *allow_type_removal=False*, *save=True*)

Updates the resource from a flat *dict*.

By flat, we mean that sub-resources are only represented by their IRIs: there is no nested sub-object structure.

This dict is supposed to be exhaustive, so absent value is removed. Some sub-resources may thus be deleted like if there were a cascade deletion.

Parameters

- **full_dict** – Flat *dict* containing the attribute values to update.
- **is_end_user** – *False* when an authorized user (not a regular end-user) wants to force some rights. Defaults to *True*. See `check_validity()` for further details.
- **allow_new_type** – If *True*, new types can be added. Please keep in mind that type change can:
  - Modify the behavior of the resource by changing its model list.
  - Interfere with the SPARQL requests using instance tests.

  If enabled, this may represent a major **security concern**. Defaults to *False*.
- **allow_type_removal** – If *True*, new types can be removed. Same security concerns than above. Defaults to *False*.
- **save** – If *True* calls `save()` after updating. Defaults to *True*.

Returns The `Resource` object itself.

full_update_from_graph(*subgraph*, *initial=False*, *is_end_user=True*, *allow_new_type=False*, *allow_type_removal=False*, *save=True*)

Similar to `full_update()` but with a RDF graph instead of a Python *dict*.

Parameters

- **subgraph** – `rdflib.Graph` object containing the full description of the resource.
- **initial** – *True* when the subgraph comes from the *data_graph* and is thus used to load `Resource` object from the triple store. Defaults to *False*.
- **is_end_user** – *False* when an authorized user (not a regular end-user) wants to force some rights. Defaults to *True*. See `check_validity()` for further details.
- **allow_new_type** – If *True*, new types can be added. Defaults to *False*. See `full_update()` for explanations about the security concerns.
- **allow_type_removal** – If *True*, new types can be removed. Same security concerns than above. Defaults to *False*.
- **save** – If *True* calls `save()` after updating. Defaults to *True*.

Returns The `Resource` object itself.

hashless_iri

Hash-less IRI of the *id* attribute. Is obtained by removing the fragment from the IRI.

id

IRI that identifies the resource.

in_same_document(*other_resource*)

Tests if two resources have the same hash-less IRI.

Returns *True* if these resources are in the same document.

is_blank_node()
Tests if *id* is a skolem IRI and should thus be considered as a blank node.

See `is_blank_node()` for further details.

Returns *True* if *id* is a locally skolemized IRI.

is_instance_of(*model*)
Tests if the resource is instance of the RDFS class of the model.

Parameters **model** – `Model` object.

Returns *True* if the resource is instance of the RDFS class.

is_valid()
Tests if the resource is valid.

Returns *False* if the resource is invalid, *True* otherwise.

classmethod **load_from_graph**(*manager*, *id*, *subgraph*, *is_new=True*)
Loads a new `Resource` object from a sub-graph.

Parameters

- **manager** – `ResourceManager` object.
- **id** – IRI of the resource.
- **subgraph** – `rdflib.Graph` object containing triples about the resource.
- **is_new** – When is *True* and *id* given, checks that the IRI is not already existing in the *union_graph*. Defaults to *True*.

Returns The `Resource` object created.

save(*is_end_user=True*)
Saves it into the *data_graph* and the *resource_cache*.

Raises an `oldman.exception.OMEditError` exception if invalid.

Parameters **is_end_user** – *False* when an authorized user (not a regular end-user) wants to force some rights. Defaults to *True*. See `check_validity()` for further details.

Returns The `Resource` object itself.

to_dict(*remove_none_values=True*, *include_different_contexts=False*, *ignored_iris=None*)
Serializes the resource into a JSON-like *dict*.

Parameters

- **remove_none_values** – If *True*, *None* values are not inserted into the dict. Defaults to *True*.
- **include_different_contexts** – If *True* local contexts are given to sub-resources. Defaults to *False*.
- **ignored_iris** – List of IRI of resources that should not be included in the *dict*. Defaults to *set()*.

Returns A *dict* describing the resource.

to_json(*remove_none_values=True*, *ignored_iris=None*)
Serializes the resource into pure JSON (not JSON-LD).

Parameters

- **remove_none_values** – If *True*, *None* values are not inserted into the dict. Defaults to *True*.
- **ignored_iris** – List of IRI of resources that should not be included in the *dict*. Defaults to *set()*.

Returns A JSON-encoded string.

to_jsonld(*remove_none_values=True*, *include_different_contexts=False*, *ignored_iris=None*)
Serializes the resource into JSON-LD.

Parameters

- **remove_none_values** – If *True*, *None* values are not inserted into the dict. Defaults to *True*.
- **include_different_contexts** – If *True* local contexts are given to sub-resources. Defaults to *False*.
- **ignored_iris** – List of IRI of resources that should not be included in the *dict*. Defaults to *set()*.

Returns A JSON-LD encoded string.

to_rdf(*rdf_format='turtle'*)
Serializes the resource into RDF.

Parameters **rdf_format** – content-type or keyword supported by RDFlib. Defaults to *"turtle"*.

Returns A string in the chosen RDF format.

types
IRI list of the RDFS classes the resource is instance of.

oldman.resource.**is_blank_node**(*iri*)
Tests if *id* is a locally skolemized IRI.

External skolemized blank nodes are not considered as blank nodes.

Parameters **iri** – IRI of the resource.

Returns *True* if is a blank node.

oldman.resource.**should_delete_resource**(*resource*)
Tests if a resource should be deleted.

Parameters **resource** – Resource object to evaluate.

Returns *True* if it should be deleted.

2.1.7 oldman.vocabulary module

oldman.vocabulary

RDF vocabulary specific to OldMan.

TODO: replace these URNs by URLs.

Parent model prioritization

In RDF, a class is often the child of multiple classes. When the code inherited from these classes (common practise in Object-Oriented Programming) is conflicting, arbitration is necessary.

In this library, we provide a RDF vocabulary to declare priorities for each parent of a given child class. A priority statement is declared as follows:

```
?cls <urn:oldman:model:ordering:hasPriority> [
    <urn:oldman:model:ordering:class> ?parent1 ;
    <urn:oldman:model:ordering:priority> 2
].
```

By default, when no priority is declared for a pair (child, parent), its priority value is set to 0.

---

oldman.vocabulary.**NEXT_NUMBER_IRI = 'urn:oldman:nextNumber'**
Used to increment IRIs.

2.1.8 Sub-packages

oldman.management package

oldman.management.ancestry module

class oldman.management.ancestry.**ClassAncestry**(*child_class_iri*, *schema_graph*)
Bases: object

Ancestry of a given RDFS class.

Parameters

- **child_class_iri** – IRI of the child RDFS class.
- **schema_graph** – rdflib.Graph object contains all the schema triples.

bottom_up
Ancestry list starting from the child.

child
Child of the ancestry.

parents(*class_iri*)
Finds the parents of a given class in the ancestry.

Parameters **class_iri** – IRI of the RDFS class.

Returns List of class IRIs

top_down
Reverse of the *bottom_up* attribute.

oldman.management.manager module

class oldman.management.manager.**ResourceManager**(*schema_graph*, *data_store*, *attr_extractor=None*, *manager_name='default'*)
Bases: object

The *resource_manager* is the central object of this OLDM.

It gives access to the DataStore object and creates Model objects. It also creates, retrieves and caches Resource objects.

Internally, it owns a ModelRegistry object.

Parameters

- **schema_graph** – `rdflib.Graph` object containing all the schema triples.
- **data_store** – `DataStore` object. Supports CRUD operations on `Resource` objects.
- **attr_extractor** – `OMAttributeExtractor` object that will extract `OMAttribute` for generating new `Model` objects. Defaults to a new instance of `OMAttributeExtractor`.
- **manager_name** – Name of this manager. Defaults to *"default"*. This name must be unique.

create(*id=None*, *types=None*, *hashless_iri=None*, ***kwargs*)

Creates a new resource and save it in the *data_store*.

See `new()` for more details.

create_model(*class_name_or_iri*, *context*, *iri_generator=None*, *iri_prefix=None*, *iri_fragment=None*, *incremental_iri=False*)

Creates a `Model` object.

To create it, they are three elements to consider:

1. Its class IRI which can be retrieved from *class_name_or_iri*;
2. Its JSON-LD context for mapping `OMAttribute` values to RDF triples;
3. The `IriGenerator` object that generates IRIs from new `Resource` objects.

The `IriGenerator` object is either:

- directly given: *iri_generator*;
- created from the parameters *iri_prefix*, *iri_fragment* and *incremental_iri*.

Parameters

- **class_name_or_iri** – IRI or JSON-LD term of a RDFS class.
- **context** – *dict*, *list* or *IRI* that represents the JSON-LD context .
- **iri_generator** – `IriGenerator` object. If given, other *iri_** parameters are ignored.
- **iri_prefix** – Prefix of generated IRIs. Defaults to *None*. If is *None* and no *iri_generator* is given, a `BlankNodeIriGenerator` is created.
- **iri_fragment** – IRI fragment that is added at the end of generated IRIs. For instance, *"me"* adds *"#me"* at the end of the new IRI. Defaults to *None*. Has no effect if *iri_prefix* is not given.
- **incremental_iri** – If *True* an `IncrementalIriGenerator` is created instead of a `RandomPrefixedIriGenerator`. Defaults to *False*. Has no effect if *iri_prefix* is not given.

Returns A new `Model` object.

data_store

`DataStore` object. Supports CRUD operations on *:class:'~oldman.resource.Resource* objects'.

declare_method(*method*, *name*, *class_iri*)

Attaches a method to the `Resource` objects that are instances of a given RDFS class.

Like in Object-Oriented Programming, this method can be overwritten by attaching a homonymous method to a class that has a higher inheritance priority (such as a sub-class).

To benefit from this method (or an overwritten one), `Resource` objects must be associated to a `Model` that corresponds to the RDFS class or to one of its subclasses.

This method can only be used before the creation of any model (except the default one).

Parameters

- **method** – Python function that takes as first argument a `Resource` object.
- **name** – Name assigned to this method.
- **class_iri** – Targetted RDFS class. If not overwritten, all the instances (`Resource` objects) should inherit this method.

filter(*types=None*, *hashless_iri=None*, *limit=None*, *eager=False*, *pre_cache_properties=None*, ***kwargs*)
See `oldman.store.datastore.DataStore.filter()`.

find_models_and_types(*type_set*)
See `oldman.management.registry.ModelRegistry.find_models_and_types()`.

get(*id=None*, *types=None*, *hashless_iri=None*, *eager_with_reversed_attributes=True*, ***kwargs*)
See `oldman.store.datastore.DataStore.get()`.

classmethod get_manager(*name*)
Gets a `ResourceManager` object by its name.

Parameters **name** – manager name.

Returns A `ResourceManager` object.

include_reversed_attributes
Is *True* if at least one of its models use some reversed attributes.

name
Name of this manager. The manager can be retrieved from its name by calling the class method `get_manager()`.

new(*id=None*, *types=None*, *hashless_iri=None*, ***kwargs*)
Creates a new `Resource` object **without saving it** in the *data_store*.

The *kwargs* dict can contains regular attribute key-values that will be assigned to `OMAttribute` objects.

Parameters

- **id** – IRI of the new resource. Defaults to *None*. If not given, the IRI is generated by the IRI generator of the main model.
- **types** – IRIs of RDFS classes the resource is instance of. Defaults to *None*. Note that these IRIs are used to find the models of the resource (see `find_models_and_types()` for more details).
- **hashless_iri** – hash-less IRI that MAY be considered when generating an IRI for the new resource. Defaults to *None*. Ignored if *id* is given.

Returns A new `Resource` object.

sparql_filter(*query*)
See `oldman.store.datastore.DataStore.sparql_filter()`.

oldman.management.registry module

class oldman.management.registry.**ModelRegistry**
Bases: `object`

A `ModelRegistry` object registers the `Model` objects.

Its main function is to find and order models from a set of class IRIs (this ordering is crucial when creating new `Resource` objects). See `find_models_and_types()` for more details.

find_models_and_types(*type_set*)
Finds the leaf models from a set of class IRIs and orders them. Also returns an ordered list of the RDFS class IRIs that come from *type_set* or were deduced from it.

Leaf model ordering is important because it determines:

1. the IRI generator to use (the one of the first model);
2. method inheritance priorities between leaf models.

Resulting orderings are cached.

Parameters **type_set** – Set of RDFS class IRIs.

Returns An ordered list of leaf `Model` objects and an ordered list of RDFS class IRIs.

get_model(*class_iri*)
Gets a `Model` object.

Parameters **class_iri** – IRI of a RDFS class

Returns A `Model` object or *None* if not found

has_specific_models()

Returns *True* if contains other models than the default one.

model_names
Names of the registered models.

register(*model*, *is_default=False*)
Registers a `Model` object.

Parameters

- **model** – the `Model` object to register.
- **is_default** – If *True*, sets the model as the default model. Defaults to *False*.

unregister(*model*)
Un-registers a `Model` object.

Parameters **model** – the `Model` object to remove from the registry.

oldman.parsing package

oldman.parsing.schema package

oldman.parsing.schema.attribute module

class oldman.parsing.schema.attribute.**OMAttributeExtractor**(*property_extractors=None*, *attr_md_extractors=None*, *use_hydra=True*, *use_jsonld_context=True*)

Bases: `object`

Extracts `OMAttribute` objects from the schema and the JSON-LD context.

Extensible in two ways:

1. New `OMPropertyExtractor` objects (new RDF vocabularies);

2.New `OMAttributeMdExtractor` objects (e.g. JSON-LD context);

3.New `ValueFormat` objects. See its `value_format_registry` attribute.

Parameters

- **property_extractors** – Defaults to *[]*.
- **attr_md_extractors** – Defaults to *[]*.
- **use_hydra** – Defaults to *True*.
- **use_jsonld_context** – Defaults to *True*.

add_attribute_md_extractor(*attr_md_extractor*)
Adds a new `OMAttributeMdExtractor` object.

add_property_extractor(*property_extractor*)
Adds a new `OMPropertyExtractor` object.

extract(*class_iri*, *type_iris*, *context_js*, *schema_graph*, *manager*)
Extracts metadata and generates `OMProperty` and `OMAttribute` objects.

Parameters

- **class_iri** – IRI of RDFS class of the future `Model` object.
- **type_iris** – Ancestry of the RDFS class.
- **context_js** – the JSON-LD context.
- **schema_graph** – `rdflib.graph.Graph` object.
- **manager** – `ResourceManager` object.

Returns *dict* of `OMAttribute` objects.

value_format_registry
`ValueFormatRegistry` object.

class oldman.parsing.schema.attribute.**ValueFormatRegistry**(*special_properties=None*, *include_default_datatypes=True*, *include_well_known_properties=True*)

Bases: `object`

Finds the `ValueFormat` object that corresponds to a `OMAttributeMetadata` object.

New `ValueFormat` objects can be added, for supporting:

1.Specific properties (eg. foaf:mbox and `EmailValueFormat`);

2.Other datatypes, as defined in the JSON-LD context or the RDFS domain or range (eg. xsd:string).

Parameters

- **special_properties** – Defaults to *{}*.
- **include_default_datatypes** – Defaults to *True*.
- **include_well_known_properties** – Defaults to *True*.

add_datatype(*datatype_iri*, *value_format*)
Registers a `ValueFormat` object for a given datatype.

Parameters

- **datatype_iri** – IRI of the datatype.
- **value_format** – `ValueFormat` object.

add_special_property(*property_iri*, *value_format*)
Registers a `ValueFormat` object for a given RDF property.

Parameters

- **property_iri** – IRI of the RDF property.
- **value_format** – `ValueFormat` object.

find_value_format(*attr_md*)
Finds the `ValueFormat` object that corresponds to a `OMAttributeMetadata` object.

Parameters **attr_md** – `OMAttributeMetadata` object.

Returns `ValueFormat` object.

oldman.parsing.schema.context module

class oldman.parsing.schema.context.**JsonLdContextAttributeMdExtractor**
Bases: `oldman.parsing.schema.context.OMAttributeMdExtractor`

`OMAttributeMdExtractor` objects that extract attribute names and datatypes from the JSON-LD context.

update(*om_properties*, *context_js*, *schema_graph*)
See `oldman.parsing.schema.context.OMAttributeMdExtractor.update()`.

class oldman.parsing.schema.context.**OMAttributeMdExtractor**
Bases: `object`

An `OMAttributeMdExtractor` object extracts `OMAttributeMetadata` tuples and transmits them to `OMProperty` objects.

update(*om_properties*, *context_js*, *schema_graph*)
Updates the `OMProperty` objects by transmitting them extracted `OMAttributeMetadata` tuples.

Parameters

- **om_properties** – *dict* of `OMProperty` objects indexed by their IRIs.
- **context_js** – JSON-LD context.
- **schema_graph** – `rdflib.graph.Graph` object.

oldman.parsing.schema.property module

class oldman.parsing.schema.property.**HydraPropertyExtractor**
Bases: `oldman.parsing.schema.property.OMPropertyExtractor`

`OMPropertyExtractor` objects that support the Hydra vocabulary.

Currently, this class supports:

- hydra:required ;
- hydra:readonly;
- hydra:writeonly .

update(*om_properties*, *class_iri*, *type_iris*, *schema_graph*, *manager*)
See `oldman.parsing.schema.property.OMPropertyExtractor.update()`.

class oldman.parsing.schema.property.**OMPropertyExtractor**
Bases: object

An OMPropertyExtractor object generates and updates OMProperty objects from the schema RDF graph.

This class is generic and must derived for supporting various RDF vocabularies.

update(*om_properties*, *class_iri*, *type_iris*, *schema_graph*, *manager*)
Generates new OMProperty objects or updates them from the schema graph.

Parameters

- **om_properties** – *dict* of OMProperty objects indexed by their IRIs and their reverse status.
- **class_iri** – IRI of RDFS class of the future Model object.
- **type_iris** – Ancestry of the RDFS class.
- **schema_graph** – rdflib.graph.Graph object.

Returns Updated *dict* OMProperty objects.

oldman.parsing.value module

class oldman.parsing.value.**AttributeValueExtractor**(*om_attribute*)
Bases: object

An AttributeValueExtractor object extracts values from RDF graphs for a given OMAttribute object.

Parameters **om_attribute** – OMAttribute object.

extract_value(*resource*, *subgraph*)
Extracts a resource attribute value from a RDF graph.

Parameters

- **resource** – Resource object.
- **subgraph** – rdflib.graph.Graph object containing the value to extract.

Returns Collection or atomic value.

oldman.rest package

oldman.rest.crud module

class oldman.rest.crud.**CRUDController**(*manager*)
Bases: object

A CRUDController object helps you to manipulate your Resource objects in a RESTful-like manner.

Please note that REST/HTTP only manipulates hash-less IRIs. A hash IRI is the combination of a hash-less IRI (fragment-less IRI) and a fragment. Multiple hashed IRIs may have the same hash-less IRI and only differ by their fragment values. This is a concern for each type of HTTP operation.

This class is generic and does not support the Collection pattern (there is no append method).

Parameters **manager** – ResourceManager object.

Possible improvements:

- Add a PATCH method.

delete(*hashless_iri*)
Deletes every `Resource` object having this hash-less IRI.

Parameters **hashless_iri** – Hash-less IRI.

get(*hashless_iri*, *content_type='text/turtle'*)
Gets the main `Resource` object having its hash-less IRI.

When multiple `Resource` objects have this hash-less IRI, one of them has to be selected. If one has no fragment value, it is selected. Otherwise, this selection is currently arbitrary.

TODO: stop selecting the resources and returns the list.

Raises an `ObjectNotFoundError` exception if no resource is found.

Parameters

- **hashless_iri** – hash-less of the resource.
- **content_type** – Content type of its representation.

Returns The selected `Resource` object.

update(*hashless_iri*, *document_content*, *content_type*, *allow_new_type=False*, *allow_type_removal=False*)
Updates every `Resource` object having this hash-less IRI.

Raises an `OMDifferentBaseIRIError` exception if tries to create of modify non-blank `Resource` objects that have a different hash-less IRI. This restriction is motivated by security concerns.

Accepts JSON, JSON-LD and RDF formats supported by RDFlib.

Parameters

- **hashless_iri** – Document IRI.
- **document_content** – Payload.
- **content_type** – Content type of the payload.
- **allow_new_type** – If *True*, new types can be added. Defaults to *False*. See `oldman.resource.Resource.full_update()` for explanations about the security concerns.
- **allow_type_removal** – If *True*, new types can be removed. Same security concerns than above. Defaults to *False*.

oldman.store package

Submodules

oldman.store.cache module

class `oldman.store.cache.`**ResourceCache**(*cache_region*)
Bases: `object`

A `ResourceCache` object caches `Resource` objects.

It interfaces a `dogpile.cache.region.CacheRegion` front-end object. If not *None*, *cache_region* must be already configured, i.e. mapped to a back-end (like Memcache or Redis). See the official list of back-ends supported by dogpile.cache.

When *cache_region* is None, no effective caching is done. However, methods `get_resource()`, `set_resource()` and `remove_resource()` can still safely be called. They just have no effect.

Parameters **cache_region** – `dogpile.cache.region.CacheRegion` object. This object must already be configured. Defaults to None (no cache).

cache_region
`dogpile.cache.region.CacheRegion` object. May be *None*.

change_cache_region(*cache_region*)
Replaces the *cache_region* attribute.

Parameters **cache_region** – `dogpile.cache.region.CacheRegion` object. May be *None*.

get_resource(*id*)
Gets a `Resource` object from the cache.

Parameters **id** – IRI of the resource.

Returns `Resource` object or *None* if not found.

invalidate_cache()
See `dogpile.cache.region.CacheRegion.invalidate()`.

Cache invalidation

Please note that this method is not supported by some `dogpile.cache.api.CacheBackend` objects. In such a case, this method has no effect so entries must be removed **explicitly** from their keys.

---

is_active()

Returns *True* if the *cache_region* is active.

remove_resource(*resource*)
Removes a `Resource` object from the cache.

Indempotent (no problem if the `Resource` object is not the cache). Does nothing if *cache_region* is *None*.

Parameters **resource** – `Resource` object to remove from the cache.

remove_resource_from_id(*id*)
`remove_resource()` is usually preferred.

Indempotent and does nothing if *cache_region* is *None*.

Parameters **id** – IRI of the resource to remove from the cache.

set_resource(*resource*)
Adds or updates a `Resource` object in the cache.

Its key is its *ìd*.

Parameters **resource** – `Resource` object to add to the cache (or update).

oldman.store.datastore module

class oldman.store.datastore.**DataStore**(*cache_region=None*)
Bases: object

A DataStore object manages CRUD operations on Resource objects.

In the future, non-CRUD operations may also be supported.

Manages the cache (ResourceCache object) of Resource object.

A ResourceManager object must be assigned after instantiation of this object.

Parameters **cache_region** – dogpile.cache.region.CacheRegion object. This object must already be configured. Defaults to None (no cache). See ResourceCache for further details.

check_and_repair_counter(*class_iri*)
Checks the counter of a given RDFS class and repairs (inits) it if needed.

Parameters **class_iri** – RDFS class IRI.

delete(*resource*, *attributes*, *former_types*)
End-users should not call it directly. Call oldman.Resource.delete() instead.

Parameters

- **resource** – Resource object.
- **attributes** – Ordered list of OMAttribute objects.
- **former_types** – List of RDFS class IRIs previously saved.

exists(*resource_iri*)
Tests if the IRI of the resource is present in the data_store.

May raise an UnsupportedDataStorageFeatureException exception.

Parameters **resource_iri** – IRI of the Resource object.

Returns *True* if exists.

filter(*types=None*, *hashless_iri=None*, *limit=None*, *eager=False*, *pre_cache_properties=None*, ***kwargs*)
Finds the Resource objects matching the given criteria.

The *kwargs* dict can contains:

1. regular attribute key-values ;
2. the special attribute *id*. If given, get() is called.

Parameters

- **types** – IRIs of the RDFS classes filtered resources must be instance of. Defaults to *None*.
- **hashless_iri** – Hash-less IRI of filtered resources. Defaults to *None*.
- **limit** – Upper bound on the number of solutions returned (e.g. SPARQL LIMIT). Positive integer. Defaults to *None*.
- **eager** – If *True* loads all the Resource objects within the minimum number of queries (e.g. one single SPARQL query). Defaults to *False* (lazy).

- **pre_cache_properties** – List of RDF ObjectProperties to pre-cache eagerly. Their values (`Resource` objects) are loaded and added to the cache. Defaults to *[]*. If given, *eager* must be *True*. Disabled if there is no cache.

Returns A generator (if lazy) or a list (if eager) of `Resource` objects.

generate_instance_number(*class_iri*)
Generates a new incremented number for a given RDFS class IRI.

May raise an `UnsupportedDataStorageFeatureException` exception.

Parameters **class_iri** – RDFS class IRI.

Returns Incremented number.

get(*id=None*, *types=None*, *hashless_iri=None*, *eager_with_reversed_attributes=True*, ***kwargs*)
Gets the first `Resource` object matching the given criteria.

The *kwargs* dict can contains regular attribute key-values.

When *id* is given, types are then checked. An `OMClassInstanceError` is raised if the resource is not instance of these classes. **Other criteria are not checked**.

Parameters

- **id** – IRI of the resource. Defaults to *None*.
- **types** – IRIs of the RDFS classes filtered resources must be instance of. Defaults to *None*.
- **hashless_iri** – Hash-less IRI of filtered resources. Defaults to *None*.
- **eager_with_reversed_attributes** – Allow to Look eagerly for reversed RDF properties. May cause some overhead for some `Resource` objects that do not have reversed attributes. Defaults to *True*.

Returns A `Resource` object or *None* if no resource has been found.

manager
The `ResourceManager` object.

Necessary for creating new `Resource` objects and accessing to `Model` objects.

reset_instance_counter(*class_iri*)
Reset the counter related to a given RDFS class.

For test purposes **only**.

Parameters **class_iri** – RDFS class IRI.

resource_cache
`ResourceCache` object.

save(*resource*, *attributes*, *former_types*)
End-users should not call it directly. Call `oldman.Resource.save()` instead.

Parameters

- **resource** – `Resource` object.
- **attributes** – Ordered list of `OMAttribute` objects.
- **former_types** – List of RDFS class IRIs previously saved.

sparql_filter(*query*)
Finds the `Resource` objects matching a given query.

Raises an `UnsupportedDataStorageFeatureException` exception if the SPARQL protocol is not supported by the concrete data_store.

Parameters **query** – SPARQL SELECT query where the first variable assigned corresponds to the IRIs of the resources that will be returned.

Returns A generator of `Resource` objects.

oldman.store.sparql module

class `oldman.store.sparql.`**SPARQLDataStore**(*data_graph*, *union_graph=None*, *cache_region=None*)

Bases: `oldman.store.datastore.DataStore`

A `SPARQLDataStore` is a `DataStore` object relying on a SPARQL 1.1 endpoint (Query and Update).

Parameters

- **data_graph** – `rdflib.graph.Graph` object where all the non-schema resources are stored by default.
- **union_graph** – Union of all the named graphs of a `rdflib.ConjunctiveGraph` or a `rdflib.Dataset`. Super-set of *data_graph* and may also include *schema_graph*. Defaults to *data_graph*. Read-only.
- **cache_region** – `dogpile.cache.region.CacheRegion` object. This object must already be configured. Defaults to None (no cache). See `ResourceCache` for further details.

check_and_repair_counter(*class_iri*)

Checks the counter of a given RDFS class and repairs (inits) it if needed.

Parameters **class_iri** – RDFS class IRI.

exists(*id*)

extract_prefixes(*other_graph*)

Adds the RDF prefix (namespace) information from an other graph to the namespace of the *data_graph*. :param other_graph: *rdflib.graph.Graph* that some prefix information.

generate_instance_number(*class_iri*)

Needed for generating incremental IRIs.

reset_instance_counter(*class_iri*)

Reset the counter related to a given RDFS class.

For test purposes **only**.

Parameters **class_iri** – RDFS class IRI.

sparql_filter(*query*)

Finds the `Resource` objects matching a given query.

Parameters **query** – SPARQL SELECT query where the first variable assigned corresponds to the IRIs of the resources that will be returned.

Returns A generator of `Resource` objects.

Module contents

oldman.utils package

oldman.utils.sparql module

oldman.utils.sparql.**build_query_part**(*verb_and_vars*, *subject_term*, *lines*)
Builds a SPARQL query.

Parameters

- **verb_and_vars** – SPARQL verb and variables.
- **subject_term** – Common subject term.
- **lines** – Lines to insert into the WHERE block.

Returns A SPARQL query.

oldman.utils.sparql.**build_update_query_part**(*verb*, *subject*, *lines*)
Builds a SPARQL Update query.

Parameters

- **verb** – SPARQL verb.
- **subject** – Common subject term.
- **lines** – Lines to insert into the WHERE block.

Returns A SPARQL Update query.

oldman.utils.sparql.**parse_graph_safely**(*graph*, **args*, ***kwargs*)
Skolemizes the input source if the graph uses a `rdflib.plugins.stores.sparqlstore.SPARQLUpdateStore` object.

Parameters

- **graph** – `rdflib.graph.Graph` object.
- **args** – Argument *list* to transmit to `rdflib.graph.Graph.parse()`.
- **kwargs** – Argument *dict* to transmit to `rdflib.graph.Graph.parse()`.

Returns The updated `rdflib.graph.Graph` object.

oldman.validation package

oldman.validation.value_format module

class oldman.validation.value_format.**AnyValueFormat**
Bases: `oldman.validation.value_format.ValueFormat`

Accepts any value.

check_value(*value*)
See `oldman.validation.value_format.ValueFormat.check_value()`.

class oldman.validation.value_format.**EmailValueFormat**
Bases: `oldman.validation.value_format.TypedValueFormat`

Checks that the value is an email address.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**HexBinaryFormat**
Bases: oldman.validation.value_format.TypedValueFormat

Checks that the value is a hexadecimal string.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

to_python(*rdf_term*)
Returns a hexstring.

class oldman.validation.value_format.**IRIValueFormat**
Bases: oldman.validation.value_format.ValueFormat

Checks that the value is an IRI.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**NegativeTypedValueFormat**(*types*)
Bases: oldman.validation.value_format.TypedValueFormat

Checks that the value is a negative number.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**NonNegativeTypedValueFormat**(*types*)
Bases: oldman.validation.value_format.TypedValueFormat

Checks that the value is a non-negative number.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**NonPositiveTypedValueFormat**(*types*)
Bases: oldman.validation.value_format.TypedValueFormat

Checks that the value is a non-positive number.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**PositiveTypedValueFormat**(*types*)
Bases: oldman.validation.value_format.TypedValueFormat

Checks that the value is a positive number.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**TypedValueFormat**(*types*)
Bases: oldman.validation.value_format.ValueFormat

Checks that the value is of a given type.

Parameters **types** – Supported Python types.

check_value(*value*)
See oldman.validation.value_format.ValueFormat.check_value().

class oldman.validation.value_format.**ValueFormat**
Bases: object

A ValueFormat object checks the values and converts *rdflib.term.Identifier* objects into Python objects.

check_value(*value*)
Raises a ValueFormatError exception if the value is wrongly formatted.

Parameters **value** – Python value to check.

to_python(*rdf_term*)
Converts a *rdflib.term.Identifier* object into a regular Python value.

By default, uses the RDFlib *toPython()* method.

Parameters **rdf_term** – *rdflib.term.Identifier* object.

Returns Regular Python object.

exception oldman.validation.value_format.**ValueFormatError**
Bases: exceptions.Exception

Invalid format detected.

modindex, *genindex* and *search*.

O

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V