yarom Documentation

Release

yarom

September 09, 2015

Contents

1 Contents **1**
- 1.1 Making data objects . . . 1
- 1.2 Adding Data to *YOUR* yarom Database . . . 2
- 1.3 Adding documentation . . . 4

2 API **7**
- 2.1 Classes . . . 7

3 Questions/concerns? **19**

4 Indices and tables **21**

Python Module Index **23**

CHAPTER 1

Contents

1.1 Making data objects

To make new objects, for the most part, you just need to make a Python class and subclass it from *yarom.dataObject.DataObject*. Say, for example, that I want to record some information about drug reactions in C. elegans. Using classes I've already created in `c_elegans.py`, I make `Drug` and `Experiment` classes in `drugRxn.py` to describe C. elegans drug reactions:

```
import yarom
from yarom import DataObject
from c_elegans import Worm, Evidence

class Drug(DataObject):
    # We set up properties in __init__
    _ = ['name']
    def defined_augment(self):
        return len(self.name.values) > 0

    def identifier_augment(self):
        return self.make_identifier_direct(self.name.values[0])

class Experiment(DataObject):
    _ = [{'name':'drug', 'type':Drug, 'multiple':False},
         {'name':'subject', 'type':Worm},
         'experimenter',
         'summary',
         'route_of_entry',
         'reaction']
```

The *defined_augment()* and *identifier_augment()* methods define the unique identifier of a drug object as being based on the drug name.

In a new file, I can then make a Drug object for moon rocks and describe an experiment by Aperture Labs:

```
import yarom
yarom.connect({"rdf.source" : "ZODB", "rdf.store_conf" : "ceDrugRxns.db"})

yarom.load_module('drugRxn')
from yarom import *

d = Drug(name='moon rocks')
d.relate('granularity', 'ground up')
```

```
e = Experiment(key='E2334', summary='C. elegans exposure to Moon rocks', experimenter='Cave Johnson')

w = Worm(generate_key=True, scientific_name="C. elegans") # the worm tested

ev = Evidence(key='ApertureLabs')
ev.relate('organization', "Aperture Labs") # Organization releasing the experimental data

e.subject(w)
e.drug(d)
e.route_of_entry('ingestion')
e.reaction('no reaction')
ev.asserts(e)
ev.save() # XXX: Don't forget this!

print_graph(config('rdf.graph'))
```

It is not necessary to put the class definitions and the objects in separate files, but the descriptions must follow a call to *yarom.connect()*.

For simple objects, this is all we have to do.

1.2 Adding Data to *YOUR* yarom Database

So, you've got some data about the and you'd like to save it with yarom, but you don't know how it's done?

You've come to the right place!

Depending on your needs you may be able to use the default classes directly to define your data. For example, you can write a little about a person:

```
import yarom as Y
Y.connect({'rdf.namespace': rdflib.namespace.Namespace("http://example.org/")})
mary = Y.DataObject(key='mary')
fido = Y.DataObject(key='fido')
mary.relate('has_pet', fido)
mary.relate('age', Y.Quantity(23, 'years'))
mary.relate('email', "mary@example.org")
```

and get the following graph (namespace prefixes omitted):

```
:DataObject a rdfs:Class .

:Relationship a rdfs:Class ;
    rdfs:subClassOf :DataObject .

rdf:Property a rdfs:Class .

rdfs:Class a rdfs:Class .

DataObject:mary a :DataObject ;
    DataObject:age "23 year"^^<http://example.org/datatypes/quantity> ;
    DataObject:email "mary@example.org" ;
    DataObject:has_pet DataObject:fido .

Relationship:subject a rdf:Property ;
    rdfs:domain :Relationship .
```

```
:SimpleProperty a rdf:Property .

rdf:type a rdf:Property ;
    rdfs:domain :DataObject ;
    rdfs:range rdfs:Class .

DataObject:fido a :DataObject .
```

Of course, this description lacks some specificity in types, and one may want to use the well-known FOAF vocabulary (while ignoring the cultural myopia that entails) for defining the age and email address of 'DataObject:mary'. To add this information in yarom, you would do something like the following:

```
FOAF = rdflib.Namespace("http://xmlns.com/foaf/0.1/")
Y.config('rdf.namespace_manager').bind('foaf', FOAF)
class Person(Y.DataObject):
    rdf_type = FOAF['Person']

class Dog(Y.DataObject):
    pass

class FOAFAge(Y.DatatypeProperty):
    link = FOAF['age']
    linkName = "foaf_age"
    owner_type = Person
    multiple = False # XXX: Default is True

class FOAFMbox(Y.UnionProperty):
    link = FOAF['mbox']
    linkName = "foaf_mbox"
    owner_type = Person # XXX: Not defining agent
    multiple = True
Y.remap()

mary = Person(key='mary')
fido = Dog(key='fido')

mary.relate('has_pet', fido)
mary.relate('age', Y.Quantity(23, 'years'), FOAFAge)
mary.relate('email', "mary@example.org", FOAFMbox)
Y.print_graph(mary.get_defined_component())
```

and get a graph like:

```
@prefix : <http://example.org/> .
@prefix Dog: <http://example.org/Dog/> .
@prefix Person: <http://example.org/Person/> .
@prefix Relationship: <http://example.org/Relationship/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:DataObject a rdfs:Class .

:Dog a rdfs:Class ;
    rdfs:subClassOf :DataObject .

:Relationship a rdfs:Class ;
    rdfs:subClassOf :DataObject .
```

```
rdf:Property a rdfs:Class .

rdfs:Class a rdfs:Class .

foaf:Person a rdfs:Class ;
    rdfs:subClassOf :DataObject .

Relationship:subject a rdf:Property ;
    rdfs:domain :Relationship .

:SimpleProperty a rdf:Property .

rdf:type a rdf:Property ;
    rdfs:domain :DataObject ;
    rdfs:range rdfs:Class .

Person:mary a foaf:Person ;
    Person:has_pet Dog:fido ;
    foaf:age "23 year"^^<http://example.org/datatypes/quantity> ;
    foaf:mbox "mary@example.org" .

foaf:mbox a rdf:Property ;
    rdfs:domain foaf:Person .

Dog:fido a :Dog .
```

More information on making new DataObject types is given in *Making data objects*.

Typically, you'll want to attach the data that you insert to entities already in the database. You can do this by specifying an to change, loading it, making additions, and saving the object:

```
mary.save()
q_person = Person()
q_person.relate('has_pet', Dog())

for p in q_person.load():
    p.relate('dog_lover', True)
    p.save()

q_person = Person()
q_person.relate('dog_lover', True)
for p in q_person.load():
    print(p) # Prints `Person:mary`
```

1.3 Adding documentation

Documentation for yarom is housed in two locations:

1. In the top-level project directory as `README.rst`.
2. As a Sphinx project under the `docs` directory

To add a page about useful facts concerning C. elegans to the documentation, include an entry in the list under `toctree` in `docs/index.rst` like:

```
worm-facts
```

and create the file `worm-facts.rst` under the `docs` directory and add a line:

```
.. _worm-facts:
```

to the top of your file, remembering to leave an empty line before adding all of your wonderful worm facts.

You can get a preview of what your documentation will look like when it is published by running `sphinx-build` on the docs directory:

```
sphinx-build -w sphinx-errors docs build_destination
```

The docs will be compiled to html which you can view by pointing your web browser at `build_destination/index.html`. If you want to view the documentation locally with the ReadTheDocs theme you'll need to download and install it.

1.3.1 API Documentation

API documentation is generated by the Sphinx autodoc extension. The format should be easy to pick up on, but a reference is available here. Just add a docstring to your function/class/method and add an `automodule` line to `PyOpenWorm/__init__.py` and your class should appear among the other documented classes.

1.3.2 Substitutions

Project-wide substitutions can be (conservatively!) added to allow for easily changing a value over all of the documentation. Currently defined substitutions can be found in `conf.py` in the `rst_epilog` setting. More about substitutions

1.3.3 Conventions

If you'd like to add a convention, list it here and start using it.

Currently there are no real conventions to follow for documentation style, but additions to the docs will be subject to style and content review by project maintainers.

CHAPTER 2

API

Most statements correspond to some action on the database. Some of these actions may be complex, but intuitively `a.B()`, the Query form, will query against the database for the value or values that are related to `a` through `B`; on the other hand, `a.B(c)`, the Update form, will add a statement to the database that `a` relates to `c` through `B`. For the Update form, a Relationship object describing the relationship stated is returned as a side-effect of the update.

The Update form can also be accessed through the set() method of a Property and the Query form through the get() method like:

```
a.B.set(c)
```

and:

```
a.B.get()
```

The get() method also allows for parameterizing the query in ways specific to the Property.

Notes:

- Of course, when these methods communicate with an external database, they may fail due to the database being unavailable and the user should be notified if a connection cannot be established in a reasonable time. Also, some objects are created by querying the database; these may be made out-of-date in that case.
- `a : {x_0,...,x_n}` means `a` could have the value of any one of `x_0` through `x_n`

2.1 Classes

class `yarom.dataObject.`**`DataObject`**(*ident=False*, *var=False*, *key=False*, *generate_key=False*, ***kwargs*)

Bases: *yarom.graphObject.GraphObject*, *yarom.dataUser.DataUser*

An object backed by the database

Attributes

rdf_type	(rdflib.term.URIRef) The RDF type URI for objects of this type
rdf_namespace	(rdflib.namespace.Namespace) The rdflib namespace (prefix for URIs) for objects from this class
properties	(list of Property) Properties belonging to this object
owner_properties	(list of Property) Properties belonging to parents of this object

__init__(*ident=False*, *var=False*, *key=False*, *generate_key=False*, ***kwargs*)
A subclass of DataObject cannot have any required positional arguments.

Parameters **ident** : rdflib.term.URIRef or str

The identifier for this DataObject

var : str

In lieu of *ident*, sets the variable for this object

key : str or object

In lieu of *ident* or *var*, sets the identifier for this DataObject using the key value. For a namespace *ex:* and key *a*, the identifier would be *ex:a*.

generate_key : bool

If true generates a random key value

kwargs : dict

Values to set for named properties

defined_augment()
This fuction must return False if *identifier_augment()* would raise an *IdentifierMissingException*. Override it when defining a non-standard identifier for subclasses of DataObjects.

get_owners(*property_name*)
Return the owners along a property pointing to this object

graph_pattern(*shorten=False*)
Get the graph pattern for this object.

It should be as simple as converting the result of triples() into a BGP

Parameters **query** : bool

Indicates whether or not the graph_pattern is to be used for querying (as in a SPARQL query) or for storage

shorten : bool

Indicates whether to shorten the URLs with the namespace manager attached to the `self`

identifier()
The identifier for this object in the rdf graph.

This identifier may be randomly generated, but an identifier returned from the graph can be used to retrieve the specific object that it refers to.

If it is desireable to customize the identifier, a subclass of DataObject should override *identifier_augment()* rather than this method.

Returns `rdflib.term.URIRef`

identifier_augment()
Override this method to define an identifier in lieu of one explicity set.

One must also override *defined_augment()* to return True whenever this method could return a valid identifier. *IdentifierMissingException* should be raised if an identifier cannot be generated by this method.

Raises **IdentifierMissingException**

classmethod open_set()
The open set contains items that must be saved directly in order for their data to be written out

retract()
Remove this object from the data store.

Retract removes an object and everything it points to, transitively, and everything which points to it.

Dual to save.

retract_object()
Remove this object from the data store.

Retract removes an object and everything it points to, transitively, and everything which points to it.

Dual to save_object.

retract_objectG()
Remove this object from the data store.

Retract removes an object and everything it points to, transitively, and everything which points to it.

Dual to save_objectG.

retract_references()
Remove all references directly to or made by this object

retract_referencesG()
Remove all references directly to or made by this object

save()
Write in-memory data to the database. Derived classes should call this to update the store.

Dual to retract.

save_object()
Write in-memory data to the database. Derived classes should call this to update the store.

Dual to retract_object.

triples()
Returns 3-tuples of the connected component of the object graph starting from this object.

Returns An iterable of triples

variable()
Returns the variable to be usedin queries with this DataObject

Raises IdentifierMissingException

defined
Returns *True* if this object has an identifier

To define a custom identifier, override *defined_augment()* to return True when your custom identifier would be defined. You must also override *identifier_augment()*

class yarom.dataObject.**ObjectCollection**(*group_name=False*, ***kwargs*)
Bases: *yarom.dataObject.DataObject*

A convenience class for working with a collection of objects

Example:

```
v = ObjectCollection('unc-13 neurons and muscles')
n = P.Neuron()
m = P.Muscle()
```

```
    n.receptor('UNC-13')
    m.receptor('UNC-13')
    for x in n.load():
        v.value(x)
    for x in m.load():
        v.value(x)
    # Save the group for later use
    v.save()
    ...
    # get the list back
    u = ObjectCollection('unc-13 neurons and muscles')
    nm = list(u.value())
```

Parameters **group_name** : string

A name of the group of objects

Attributes

name	(DatatypeProperty) The name of the group of objects
group_name	(DataObject) an alias for `name`
member	(ObjectProperty) An object in the group
add	(ObjectProperty) an alias for `value`

class `yarom.dataObject.`**`PropertyDataObject`**(*ident=False*, *var=False*, *key=False*, *generate_key=False*, ***kwargs*)

Bases: *yarom.dataObject.DataObject*

A PropertyDataObject represents the property-as-object.

Try not to confuse this with the Property class

class `yarom.dataObject.`**`RDFProperty`**

Bases: `yarom.dataObject.DataObjectSingleton`

The DataObject corresponding to rdf:Property

class `yarom.dataObject.`**`RDFSClass`**

Bases: `yarom.dataObject.DataObjectSingleton`

The DataObject corresponding to rdfs:Class

`yarom.dataObject.`**`validate`**(*do_type*)

Given a DataObject type, call validate() on all objects of that type in the RDF object graph

`yarom.dataObject.`**`validateG`**(*do_type*)

Given a DataObject type, call validate() on all objects of that type in the Python object graph

class `yarom.dataUser.`**`DataUser`**(***kwargs*)

Bases: *yarom.configure.Configureable*

A convenience wrapper for users of the database

Classes which use the database should inherit from DataUser.

`add_statements`(*graph*)

Add a set of statements to the database.

Annotates the addition with uploader name, etc

Parameters **triples** : iter of (`rdflib.term.URIRef`, `rdflib.term.URIRef`, `rdflib.term.URIRef`)

A set of triples to add to the graph

`retract_statements`(*statements*)

Remove a set of statements from the database.

Parameters **triples** : iter of (`rdflib.term.URIRef`, `rdflib.term.URIRef`, `rdflib.term.URIRef`)

A set of triples to remove

class `yarom.data.`**`Data`**(*conf=False*)

Bases: *yarom.configure.Configuration*, *yarom.configure.Configureable*

Provides configuration for access to the database.

Usally doesn't need to be accessed directly

`closeDatabase`()

Close a the configured database

classmethod **`open`**(*file_name*)

Open a file storing configuration in a JSON format

`openDatabase`()

Open a the configured database

class `yarom.data.`**`RDFSource`**(***kwargs*)

Bases: *yarom.configure.ConfigValue*, *yarom.configure.Configureable*

Base class for data sources.

Alternative sources should dervie from this class

`open`()

Called on `yarom.connect()` to set up and return the rdflib graph. Must be overridden by sub-classes.

class `yarom.data.`**`SerializationSource`**(***kwargs*)

Bases: *yarom.data.RDFSource*

Reads from an RDF serialization or, if the configured database is more recent, then from that.

The database store is configured with:

```
"rdf.source" = "serialization"
"rdf.store" = <your rdflib store name here>
"rdf.serialization" = <your RDF serialization>
"rdf.serialization_format" = <your rdflib serialization format used>
"rdf.store_conf" = <your rdflib store configuration here>
```

class `yarom.data.`**`TrixSource`**(***kwargs*)

Bases: *yarom.data.SerializationSource*

A SerializationSource specialized for TriX

The database store is configured with:

```
"rdf.source" = "trix"
"rdf.trix_location" = <location of the TriX file>
"rdf.store" = <your rdflib store name here>
"rdf.store_conf" = <your rdflib store configuration here>
```

class yarom.data.**SPARQLSource**(**kwargs*)
Bases: *yarom.data.RDFSource*

Reads from and queries against a remote data store

```
"rdf.source" = "sparql_endpoint"
```

class yarom.data.**SleepyCatSource**(**kwargs*)
Bases: *yarom.data.RDFSource*

Reads from and queries against a local Sleepycat database

The database can be configured like:

```
"rdf.source" = "Sleepycat"
"rdf.store_conf" = <your database location here>
```

class yarom.data.**DefaultSource**(**kwargs*)
Bases: *yarom.data.RDFSource*

Reads from and queries against a configured database.

The default configuration.

The database store is configured with:

```
"rdf.source" = "default"
"rdf.store" = <your rdflib store name here>
"rdf.store_conf" = <your rdflib store configuration here>
```

Leaving unconfigured simply gives an in-memory data store.

class yarom.data.**ZODBSource**(*args*, **kwargs*)
Bases: *yarom.data.RDFSource*

Reads from and queries against a configured Zope Object Database.

If the configured database does not exist, it is created.

The database store is configured with:

```
"rdf.source" = "ZODB"
"rdf.store_conf" = <location of your ZODB database>
```

Leaving unconfigured simply gives an in-memory data store.

exception yarom.configure.**BadConf**
Bases: Exception

Special exception subclass for alerting the user to a bad configuration

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

class yarom.configure.**ConfigValue**
Bases: object

A value to be configured.

Elements of a *Configuration* are, in fact, *ConfigValue* objects. They can be resolved an arbitrary time after the *Configuration* object is created by calling *get()*.

get()
Override this method to return a value when a configuration variable is accessed

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

class yarom.configure.**Configuration**(***kwargs*)
Bases: object

A simple configuration object. Enables setting and getting key-value pairs

copy(*other*)
Copy configuration values from a different object.

Parameters **other** : dict or Configuration

A dict or Configuration object to copy the configuration from

Returns self

get(*pname*, *default=None*)
Retrieve a configuration value.

Parameters **pname** : str

The key of the value to return.

default : object

The value to return if there is no value corresponding to the given key

Returns object

The value corresponding to the key in pname or *default* if none is available and a default is provided.

Raises **KeyError**

If the given key has no associated value and no default is provided

link(**names*)
Call link() with the names of configuration values that should always be the same to link them together

classmethod **open**(*file_name*)
Open a configuration file and read it to build the internal state.

Parameters **file_name** : str

The name of a configuration file encoded as JSON

Returns Configuration

a Configuration object with the configuration taken from the JSON file

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

class yarom.configure.**Configureable**(*conf=None*)
Bases: object

An object which can be configured.

A Configureable object can access a *Configuration* object, Configureable.conf, which is shared among all Configureable objects.

The configuration variables which can affect the behavior of a class should be documented in the configuration_variables class variable. This table will be checked on each access of Configureable.conf

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

conf = <yarom.data.Data object>
The configuration

configuration_variables = {}
A table of configuration values used by the Configureable object for the purpose of documentation.

The table is indexed by the configuration value. Among the data included in the table should be:

- a "description" which describes how the configuration value is used *within the configureable object*: broad generalization about the variable shouldn't be here.
- a "type" for the value of the config may also be included and may be a Python `type` or just a string description. This isn't at all intended to be used for type checking, but *is purely descriptive*.
- a "directly_configureable" indicator which should be set to `True` if the value passed in to the object for configuration variable is used more-or- less directly by the object. Sanitization of the value or translation into a more specific form are acceptable for a variable that is nonetheless directly_configureable. On the other hand, a configuration variable that has its value set within the object should have directly_configureable set to `False`.

exception yarom.graphObject.**IdentifierMissingException**(*dataObject='[unspecified object]'*, **args*, ***kwargs*)
Bases: Exception

Indicates that an identifier should be available for the object in question, but there is none

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

class yarom.graphObject.**GraphObject**(***kwargs*)
Bases: object

An object which can be included in the object graph.

An abstract base class.

identifier()
Must return an object representing this object or else raise an Exception.

variable()
Must return a `Variable` object that identifies this *GraphObject* in queries.

The variable can be randomly generated when the object is created and stored in the object.

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

defined
Returns true if an *identifier()* would return an identifier

class yarom.graphObject.**GraphObjectQuerier**(*q*, *graph*)
Bases: object

Performs queries for objects in the given graph.

The querier queries for objects at the center of a star graph. In SPARQL, the query has the form:

```
SELECT ?x WHERE {
    ?x  <p1> ?o1 .
    ?o1 <p2> ?o2 .
     ...
    ?on <pn> <a> .
```

```
        ?x  <q1> ?n1 .
        ?n1 <q2> ?n2 .
         ...
        ?nn <qn> <b> .
    }
```

It is allowed that `<px> == <py>` for `x != y`.

Queries such as:

```
    SELECT ?x WHERE {
        ?x  <p1> ?o1 .
         ...
        ?on <pn>  ?y .
    }
```

or:

```
    SELECT ?x WHERE {
        ?x  <p1> ?o1 .
         ...
        ?on <pn>  ?x .
    }
```

or:

```
    SELECT ?x WHERE {
        ?x  ?z ?o .
    }
```

or:

```
    SELECT ?x WHERE {
        ?x  ?z <a> .
    }
```

are not supported and will be ignored without error.

__init__(*q*, *graph*)
Initialize the querier.

Call the GraphObjectQuerier object to perform the query.

Parameters **q** : *GraphObject*

The object which is queried on

graph : object

The graph from which the objects are queried. Must implement a method `triples()` that takes a triple pattern, `t`, and returns a set of triples matching that pattern. The pattern for `t` is `t[i] = None`, 0 <= i <= 2, indicates that the i'th position can take any value.

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

class yarom.graphObject.**ComponentTripler**(*start*)
Bases: object

Gets a set of triples with given graph object in the first or last position.

The ComponentTripler does not query against a backing graph, but instead uses the properties attached to the object.

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

class yarom.mapper.**MappedClass**(*name*, *bases*, *dct*)
Bases: type

A type for MappedClasses

Sets up the graph with things needed for MappedClasses

deregister()
Removes the class from the object graph.

Should make it possible to garbage collect

deregister() never touches the RDF graph itself.

classmethod make_class(*name*, *bases*, *objectProperties=False*, *datatypeProperties=False*)
Intended to be used for setting up a class from the RDF graph, for instance.

map()
Maps the class to the configured rdf graph.

register()
Sets up the object graph related to this class

regsister() never touches the RDF graph itself.

Also registers the properties of this DataObject

unmap()
Unmaps the class

yarom.mapper.**get_most_specific_rdf_type**(*types*)
Gets the most specific rdf_type.

Returns the URI corresponding to the lowest in the DataObject class hierarchy from among the given URIs.

yarom.mapper.**oid**(*identifier_or_rdf_type*, *rdf_type=False*)
Create an object from its rdf type

Parameters **identifier_or_rdf_type** : str or rdflib.term.URIRef

If *rdf_type* is provided, then this value is used as the identifier for the newly created object. Otherwise, this value will be the rdf_type of the object used to determine the Python type and the object's identifier will be randomly generated.

rdf_type : str, rdflib.term.URIRef, False

If provided, this will be the rdf_type of the newly created object.

Returns The newly created object

yarom.**connect**(*conf=False*, *do_logging=False*, *data=False*, *dataFormat='n3'*)
Load desired configuration and open the database

Parameters **conf** : str, *Data*, *Configuration* or dict, optional

The configuration for the YAROM connection

do_logging : bool, optional

If True, turn on debug level logging. The default is False.

data : str, optional

If provided, specifies a file to load into the library.

dataFormat : str, optional

If provided, specifies the file format of the file pointed specified by *data*.

The formats available are those accepted by RDFLib's serializer plugins. 'n3' is the default.

yarom.**disconnect**(*c=False*)
Close the database

yarom.**loadConfig**(*f*)
Load configuration for the module

yarom.**setConf**(*conf*)
Set the configuration

Parameters **conf** : str, Data, Configuration or dict, optional

The configuration to load.

If a Data object is provided, then it's used as is for the configuration. If either a Python dict or a Configuration object are provided, then the contents of that object is used to make a Data object for configuration. If a string is provided, then the file is read in as JSON to be parsed as a dict and from there is treated as if you had passed that dict to connect.

The default action is to attempt to open a file called 'yarom.conf' from your current directory as the configuration. Failing that, an 'empty' config with default values will be loaded.

CHAPTER 3

Questions/concerns?

Bug reports and questions can be posted to the issue tracker on Github.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

y

yarom, 7
yarom.configure, 12
yarom.data, 11
yarom.dataObject, 7
yarom.dataUser, 10
yarom.graphObject, 14
yarom.mapper, 16

Symbols

__init__() (yarom.dataObject.DataObject method), 7
__init__() (yarom.graphObject.GraphObjectQuerier method), 15
__weakref__ (yarom.configure.BadConf attribute), 12
__weakref__ (yarom.configure.ConfigValue attribute), 12
__weakref__ (yarom.configure.Configuration attribute), 13
__weakref__ (yarom.configure.Configureable attribute), 13
__weakref__ (yarom.graphObject.ComponentTripler attribute), 16
__weakref__ (yarom.graphObject.GraphObject attribute), 14
__weakref__ (yarom.graphObject.GraphObjectQuerier attribute), 15
__weakref__ (yarom.graphObject.IdentifierMissingException attribute), 14

A

add_statements() (yarom.dataUser.DataUser method), 10

B

BadConf, 12

C

closeDatabase() (yarom.data.Data method), 11
ComponentTripler (class in yarom.graphObject), 15
conf (yarom.configure.Configureable attribute), 14
Configuration (class in yarom.configure), 13
configuration_variables (yarom.configure.Configureable attribute), 14
Configureable (class in yarom.configure), 13
ConfigValue (class in yarom.configure), 12
connect() (in module yarom), 16
copy() (yarom.configure.Configuration method), 13

D

Data (class in yarom.data), 11
DataObject (class in yarom.dataObject), 7
DataUser (class in yarom.dataUser), 10
DefaultSource (class in yarom.data), 12
defined (yarom.dataObject.DataObject attribute), 9
defined (yarom.graphObject.GraphObject attribute), 14
defined_augment() (yarom.dataObject.DataObject method), 8
deregister() (yarom.mapper.MappedClass method), 16
disconnect() (in module yarom), 17

G

get() (yarom.configure.Configuration method), 13
get() (yarom.configure.ConfigValue method), 12
get_most_specific_rdf_type() (in module yarom.mapper), 16
get_owners() (yarom.dataObject.DataObject method), 8
graph_pattern() (yarom.dataObject.DataObject method), 8
GraphObject (class in yarom.graphObject), 14
GraphObjectQuerier (class in yarom.graphObject), 14

I

identifier() (yarom.dataObject.DataObject method), 8
identifier() (yarom.graphObject.GraphObject method), 14
identifier_augment() (yarom.dataObject.DataObject method), 8
IdentifierMissingException, 14

L

link() (yarom.configure.Configuration method), 13
loadConfig() (in module yarom), 17

M

make_class() (yarom.mapper.MappedClass class method), 16
map() (yarom.mapper.MappedClass method), 16
MappedClass (class in yarom.mapper), 16

O

ObjectCollection (class in yarom.dataObject), 9
oid() (in module yarom.mapper), 16

open() (yarom.configure.Configuration class method), 13
open() (yarom.data.Data class method), 11
open() (yarom.data.RDFSource method), 11
open_set() (yarom.dataObject.DataObject class method), 8
openDatabase() (yarom.data.Data method), 11

P

PropertyDataObject (class in yarom.dataObject), 10

R

RDFProperty (class in yarom.dataObject), 10
RDFSClass (class in yarom.dataObject), 10
RDFSource (class in yarom.data), 11
register() (yarom.mapper.MappedClass method), 16
retract() (yarom.dataObject.DataObject method), 9
retract_object() (yarom.dataObject.DataObject method), 9
retract_objectG() (yarom.dataObject.DataObject method), 9
retract_references() (yarom.dataObject.DataObject method), 9
retract_referencesG() (yarom.dataObject.DataObject method), 9
retract_statements() (yarom.dataUser.DataUser method), 11

S

save() (yarom.dataObject.DataObject method), 9
save_object() (yarom.dataObject.DataObject method), 9
SerializationSource (class in yarom.data), 11
setConf() (in module yarom), 17
SleepyCatSource (class in yarom.data), 12
SPARQLSource (class in yarom.data), 11

T

triples() (yarom.dataObject.DataObject method), 9
TrixSource (class in yarom.data), 11

U

unmap() (yarom.mapper.MappedClass method), 16

V

validate() (in module yarom.dataObject), 10
validateG() (in module yarom.dataObject), 10
variable() (yarom.dataObject.DataObject method), 9
variable() (yarom.graphObject.GraphObject method), 14

Y

yarom (module), 7
yarom.configure (module), 12
yarom.data (module), 11
yarom.dataObject (module), 7
yarom.dataUser (module), 10
yarom.graphObject (module), 14
yarom.mapper (module), 16

Z

ZODBSource (class in yarom.data), 12