PyOpenWorm Documentation

Release alpha0.5

PyOpenWorm

Jun 06, 2018

Contents

1 PyOpenWorm **3**
- 1.1 PyOpenWorm package . . . 3

2 For Users **39**
- 2.1 PyOpenWorm Data Sources . . . 39
- 2.2 Requirements for data storage in OpenWorm . . . 41
- 2.3 Adding Data to *YOUR* OpenWorm Database . . . 43
- 2.4 Making data objects . . . 46
- 2.5 Sharing Data with other users . . . 47

3 For Developers **49**
- 3.1 Testing in PyOpenWorm . . . 49
- 3.2 Adding documentation . . . 49
- 3.3 RDF semantics for PyOpenWorm . . . 50
- 3.4 RDF structure for PyOpenWorm . . . 51
- 3.5 Population() . . . 52
- 3.6 NeuroML() . . . 52
- 3.7 PyOpenWorm coding standards . . . 52

4 Issues **53**

5 Indices and tables **55**

Python Module Index **57**

Our main README is available online on Github.[1] This documentation contains additional materials beyond what is covered there.

Contents:

[1] http://github.com/openworm/PyOpenWorm

CHAPTER 1

PyOpenWorm

1.1 PyOpenWorm package

1.1.1 PyOpenWorm

OpenWorm Unified Data Abstract Layer.

An introduction to PyOpenWorm can be found in the README on our Github page.

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.

Relationship objects are key to the *`Evidence class`* for sourcing statements. Relationships can themselves be members in a relationship, allowing for fairly complex hierarchical statements to be made about entities.

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 Also, some objects are created by querying the database; these may be made out-of-date in that case.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`

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

PyOpenWorm.**loadData**(*data='OpenWormData/WormData.n3'*, *dataFormat='n3'*, *skipIfNewer=False*)
Load data into the underlying database of this library.

XXX: This is only guaranteed to work with the ZODB database.

Parameters

- **data** – (Optional) Specify the file to load into the library
- **dataFormat** – (Optional) Specify the file format to load into the library. Currently n3 is supported
- **skipIfNewer** – (Optional) Skips loading of data if the database file is newer than the data to be loaded in. This is determined by the modified time on the main database file compared to the modified time on the data file.

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

PyOpenWorm.**connect**(*configFile=False*, *conf=False*, *do_logging=False*, *data=False*, *dataFormat='n3'*)

Load desired configuration and open the database

Parameters

- **configFile** – (Optional) The configuration file for PyOpenWorm
- **conf** – (Optional) If true, initializes a data object with the PyOpenWorm configuration
- **do_logging** – (Optional) If true, turn on debug level logging
- **data** – (Optional) If provided, specify the file to load into the library
- **dataFormat** – (Optional) If provided, specify the file format to load into the library. Currently n3 is supported

PyOpenWorm.**config**(*key=None*)
Gets the main configuration for the whole PyOpenWorm library.

Returns the instance of the Configure class currently operating.

1.1.2 Subpackages

PyOpenWorm.data_trans package

Adding for relative imports

Submodules

PyOpenWorm.data_trans.common_data module

PyOpenWorm.data_trans.connections module

class PyOpenWorm.data_trans.connections.**ConnectomeCSVDataSource**(***kwargs*)
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.connections.**NeuronConnectomeCSVTranslation**(***kwargs*)
Bases: *PyOpenWorm.datasource.GenericTranslation*

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

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

class PyOpenWorm.data_trans.connections.**NeuronConnectomeCSVTranslator**
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.connections.ConnectomeCSVDataSource*, *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/NeuronConnectomeCSVTranslator

output_type
alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translation_type
alias of *NeuronConnectomeCSVTranslation*

make_translation(*sources*)
It's intended that implementations of DataTranslator will override this method to make custom Translations according with how different arguments to Translate are (or are not) distinguished.

The actual properties of a Translation subclass must be defined within the 'translate' method

translate(*data_source*, *neurons_source*, *muscles_source*)
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

PyOpenWorm.data_trans.csv_ds module

class PyOpenWorm.data_trans.csv_ds.**CSVDataSource**(***kwargs*)

Bases: *PyOpenWorm.data_trans.local_file_ds.LocalFileDataSource*

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.csv_ds.**CSVDataTranslator**

Bases: *PyOpenWorm.datasource.DataTranslator*

Input type(s): *PyOpenWorm.datasource.DataSource*

Output type(s): *PyOpenWorm.datasource.DataSource*

URI: None

PyOpenWorm.data_trans.data_with_evidence_ds module

class PyOpenWorm.data_trans.data_with_evidence_ds.**DataWithEvidenceDataSource**(**args*, ***kwargs*)

Bases: *PyOpenWorm.datasource.DataSource*

Evidence context [DatatypeProperty] Attribute: *evidence_context*

The context in which evidence for the "Data context" is defined

Data context [DatatypeProperty] Attribute: *data_context*

The context in which primary data for this data source is defined

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

PyOpenWorm.data_trans.local_file_ds module

class PyOpenWorm.data_trans.local_file_ds.**LocalFileDataSource**(***kwargs*)

Bases: *PyOpenWorm.datasource.DataSource*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

PyOpenWorm.data_trans.neuron_data module

class PyOpenWorm.data_trans.neuron_data.**NeuronCSVDataSource**(***kwargs*)

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

BibTeX files [DatatypeProperty] Attribute: *bibtex_files*

List of BibTeX files that are referenced in the csv file by entry ID

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.neuron_data.**NeuronCSVDataTranslator**(**args*, ***kwargs*)

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.neuron_data.NeuronCSVDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/NeuronCSVDataTranslator

input_type
alias of *NeuronCSVDataSource*

output_type
alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translate(*data_source*)
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

PyOpenWorm.data_trans.wormatlas module

class PyOpenWorm.data_trans.wormatlas.**WormAtlasCellListDataSource**(***kwargs*)
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.wormatlas.**WormAtlasCellListDataTranslation**(***kwargs*)
Bases: *PyOpenWorm.datasource.GenericTranslation*

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

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

class PyOpenWorm.data_trans.wormatlas.**WormAtlasCellListDataTranslator**
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataSource*, *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/WormAtlasCellListDataTranslator

output_type
alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translation_type
alias of *WormAtlasCellListDataTranslation*

make_translation(*sources*)
It's intended that implementations of DataTranslator will override this method to make custom Translations according with how different arguments to Translate are (or are not) distinguished.

The actual properties of a Translation subclass must be defined within the 'translate' method

translate(*data_source*, *neurons_source*)
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

PyOpenWorm.data_trans.wormbase module

class PyOpenWorm.data_trans.wormbase.**MuscleWormBaseCSVTranslator**
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.wormbase.WormBaseCSVDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/MuscleWormBaseCSVTranslator

input_type
alias of *WormBaseCSVDataSource*

output_type
alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translate(*data_source*)
Upload muscles and the neurons that connect to them

class PyOpenWorm.data_trans.wormbase.**NeuronWormBaseCSVTranslator**
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.wormbase.WormBaseCSVDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/NeuronWormBaseCSVTranslator

input_type
alias of *WormBaseCSVDataSource*

output_type
alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translate(*data_source*)
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

class PyOpenWorm.data_trans.wormbase.**WormBaseCSVDataSource**(***kwargs*)
Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.wormbase.**WormbaseIonChannelCSVDataSource**(***kwargs*)

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.wormbase.**WormbaseIonChannelCSVTranslator**

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.wormbase.WormbaseIonChannelCSVDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/WormbaseIonChannelCSVTranslator

input_type
alias of *WormbaseIonChannelCSVDataSource*

output_type
alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translate(*data_source*)
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

class PyOpenWorm.data_trans.wormbase.**WormbaseTextMatchCSVDataSource**(*cell_type*, *initial_cell_column*, ***kwargs*)

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

CSV file name [DatatypeProperty] Attribute: *csv_file_name*

Header column names [DatatypeProperty] Attribute: *csv_header*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

File name [DatatypeProperty] Attribute: *file_name*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.data_trans.wormbase.**WormbaseTextMatchCSVTranslator**

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataTranslator*

Input type(s): *PyOpenWorm.data_trans.wormbase.WormbaseTextMatchCSVDataSource*

Output type(s): *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

URI: http://openworm.org/entities/translators/WormbaseTextMatchCSVTranslator

input_type

alias of *WormbaseTextMatchCSVDataSource*

output_type

alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translate(*data_source*)

Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

1.1.3 Submodules

PyOpenWorm.bibtex module

class PyOpenWorm.bibtex.**BibTexDataSource**(*bibtex_file_name*, ***kwargs*)

Bases: *PyOpenWorm.datasource.DataSource*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.bibtex.**BibTexDataTranslator**

Bases: *PyOpenWorm.datasource.DataTranslator*

Input type(s): *PyOpenWorm.datasource.DataSource*

Output type(s): *PyOpenWorm.datasource.DataSource*

URI: None

data_source_type
alias of *BibTexDataSource*

translate()
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

PyOpenWorm.bibtex.**author**(*record*)
Split author field by 'and' into a list of names.

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

PyOpenWorm.bibtex.**bibtex_to_document**(*bibtex_entry*, *context=None*)
Takes a single BibTeX entry and translates it into a Document object

PyOpenWorm.bibtex.**customizations**(*record*)
Use some functions delivered by the library

Parameters **record** – a record

Returns – customized record

PyOpenWorm.bibtex.**doi**(*record*)

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

PyOpenWorm.biology module

class PyOpenWorm.biology.**BiologyType**(***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

PyOpenWorm.cell module

class PyOpenWorm.cell.**Cell**(*name=None*, *lineageName=None*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

A biological cell.

All cells with the same name are considered to be the same object.

Parameters

name [str] The name of the cell

lineageName [str] The lineageName of the cell

blast()
Return the blast name.

Example:

```
>>> c = Cell(name="ADAL")
>>> c.blast() # Returns "AB"
```

Note that this isn't a Property. It returns the blast extracted from the ''first" lineageName saved.

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.

identifier_augment(**args*, ***kwargs*)
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

description
A description of the cell

divisionVolume
The volume of the cell at division

Example:

```
>>> v = Quantity("600","(um)^3")
>>> c = Cell(lineageName="AB plapaaaap")
>>> c.divisionVolume(v)
```

lineageName
The lineageName of the cell Example:

```
>>> c = Cell(name="ADAL")
>>> c.lineageName() # Returns ["AB plapaaaapp"]
```

name
The 'adult' name of the cell typically used by biologists when discussing C. elegans

PyOpenWorm.channel module

class PyOpenWorm.channel.**Channel**(*name=False*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

A biological ion channel.

Attributes

Models [Property] Get experimental models of this ion channel

subfamily [DatatypeProperty] Ion channel's subfamily

name [DatatypeProperty] Ion channel's name

description [DatatypeProperty] A description of the ion channel

gene_name [DatatypeProperty] Name of the gene that codes for this ion channel

gene_WB_ID [DatatypeProperty] Wormbase ID of the encoding gene

gene_class [DatatypeProperty] Classification of the encoding gene

proteins [DatatypeProperty] Proteins associated with this channel

expression_pattern [ObjectProperty]

class PyOpenWorm.channel.**ExpressionPattern**(*wormbaseID=None*, *description=None*, ***kwargs*)

Bases: *PyOpenWorm.biology.BiologyType*

PyOpenWorm.channelworm module

class PyOpenWorm.channelworm.**ChannelModel**(*modelType=False*, **args*, ***kwargs*)

Bases: *PyOpenWorm.dataObject.DataObject*

A model for an ion channel.

There may be multiple models for a single channel.

Parameters

modelType [DatatypeProperty] What this model is based on (either "homology" or "patch-clamp")

Attributes

modelType [DatatypeProperty] Passed in on construction

ion [DatatypeProperty] The type of ion this channel selects for

gating [DatatypeProperty] The gating mechanism for this channel ("voltage" or name of ligand(s))

references [Property] Evidence for this model. May be either Experiment or Evidence object(s).

conductance [DatatypeProperty] The conductance of this ion channel. This is the initial value, and should be entered as a Quantity object.

Example usage:: # Create a ChannelModel >>> cm = P.ChannelModel() # Create Evidence object >>> ev = P.Evidence(author='White et al.', date='1986') # Assert >>> ev.asserts(cm) >>> ev.save()

class PyOpenWorm.channelworm.**HomologyChannelModel**(***kwargs*)

Bases: *PyOpenWorm.channelworm.ChannelModel*

class PyOpenWorm.channelworm.**PatchClampChannelModel**(***kwargs*)

Bases: *PyOpenWorm.channelworm.ChannelModel*

class PyOpenWorm.channelworm.**PatchClampExperiment**(*reference=False*, ***kwargs*)

Bases: *PyOpenWorm.experiment.Experiment*

Store experimental conditions for a patch clamp experiment.

Attributes

Ca_concentration [DatatypeProperty] Calcium concentration

Cl_concentration [DatatypeProperty] Chlorine concentration

blockers [DatatypeProperty] Channel blockers used for this experiment

cell [DatatypeProperty] The cell this experiment was performed on

cell_age [DatatypeProperty] Age of the cell

delta_t [DatatypeProperty]

duration [DatatypeProperty]

end_time [DatatypeProperty]

extra_solution [DatatypeProperty]

initial_voltage [DatatypeProperty] Starting voltage of the patch clamp

ion_channel [DatatypeProperty] The ion channel being clamped

membrane_capacitance [DatatypeProperty] Initial membrane capacitance

mutants [DatatypeProperty] Type(s) of mutants being used in this experiment

patch_type [DatatypeProperty] Type of patch clamp being used ('voltage' or 'current')

pipette_solution [DatatypeProperty] Type of solution in the pipette

protocol_end [DatatypeProperty]

protocol_start [DatatypeProperty]

protocol_step [DatatypeProperty]

start_time [DatatypeProperty]

temperature [DatatypeProperty]

type [DatatypeProperty]

PyOpenWorm.command module

exception PyOpenWorm.command.**InvalidGraphException**
Bases: exceptions.Exception

exception PyOpenWorm.command.**UnreadableGraphException**
Bases: exceptions.Exception

class PyOpenWorm.command.**POW**
Bases: object

Attributes set on this object parameterize commands, but they are not to be changed by the commands

add_graph(*url=None*, *context=None*, *include_imports=True*)
Fetch a graph and add it to the local store.

Parameters

url [str] The URL of the graph to fetch

context [rdflib.term.URIRef] If provided, only this context and, optionally, its imported graphs will be added.

include_imports [bool] If True, imports of the named context will be included. Has no effect if context is None.

clone(*url=None*, *update_existing_config=False*)
Clone a data store

Parameters

url [str] URL of the data store to clone

commit()

diff()

fetch_graph(*url*)
Fetch a graph

init(*update_existing_config=False*)
Makes a new graph store.

The configuration file will be created if it does not exist. If it *does* exist, the location of the database store will, by default, not be changed in that file

Parameters

update_existing_config [bool] If True, updates the existing config file to point to the given file for the store configuration

merge()

push()

reconstitute(*data_source*)
Recreate a data source by executing the chain of translators that went into making it.

tag()

translate(*translator*, *output_key=None*, **data_sources*, ***named_data_sources*)
Do a translation with the named translator and inputs

config_file = None
The config file name

graph_accessor_finder = None
Callable that returns a graph accessor when given a URL for the graph

store_name = None
The file name of the database store

PyOpenWorm.configure module

exception PyOpenWorm.configure.**BadConf**
Bases: exceptions.Exception

Special exception subclass for alerting the user to a bad configuration

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

A value to be configured. Base class intended to be subclassed, as its only method is not implemented

class PyOpenWorm.configure.**Configure**(***initial_values*)
Bases: object

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

copy(*other*)
Copy this configuration into a different object.

Parameters **other** – A different configuration object to copy the configuration from this object into

Returns

get(*pname*, *default=<object object>*)
Get some parameter value out by asking for a key

Parameters

- **pname** – they key of the value you want to return.
- **default** – True if you want the default value, False if you don't (default)

Returns The value corresponding to the key in pname

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** – configuration file encoded as JSON

Returns a Configure object with the configuration taken from the JSON file

class PyOpenWorm.configure.**Configureable**(*conf=None*, ***kwargs*)
Bases: object

An object which can accept configuration. A base class intended to be subclassed.

get(*pname*, *default=None*)
The getter for the configuration

Parameters

- **pname** – The key to retreive the value of interest
- **default** – True if you want the default value, False if you don't (default)

Returns Returns the configuration value corresponding to the key pname.

PyOpenWorm.connection module

class PyOpenWorm.connection.**Connection**(*pre_cell=None*, *post_cell=None*, *number=None*, *syntype=None*, *synclass=None*, *termination=None*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

Connection between Cells

Parameters

pre_cell [string, Muscle or Neuron, optional] The pre-synaptic cell

post_cell [string, Muscle or Neuron, optional] The post-synaptic cell

number [int, optional] The weight of the connection

syntype [{'gapJunction', 'send'}, optional] The kind of synaptic connection. 'gapJunction' indicates a gap junction and 'send' a chemical synapse

synclass [string, optional] The kind of Neurotransmitter (if any) sent between *pre_cell* and *post_cell*

Attributes

termination [{'neuron', 'muscle'}] Where the connection terminates. Inferred from type of post_cell

PyOpenWorm.context module

class PyOpenWorm.context.**Context**(*key=None*, *imported=()*, *ident=None*, *mapper=None*, *base_namespace=None*, ***kwargs*)
Bases: *PyOpenWorm.import_contextualizer.ImportContextualizer*, *PyOpenWorm.contextualize.Contextualizable*, *PyOpenWorm.data.DataUser*

A context. Analogous to an RDF context, with some special sauce

class PyOpenWorm.context.**ContextContextManager**(*ctx*, *to_import*)
Bases: object

The context manager created when Context::__call__ is passed a dict

class PyOpenWorm.context.**ContextMeta**
Bases: *PyOpenWorm.contextualize.ContextualizableClass*

class PyOpenWorm.context.**QueryContext**(*graph*, **args*, ***kwargs*)
Bases: *PyOpenWorm.context.Context*

PyOpenWorm.contextDataObject module

class PyOpenWorm.contextDataObject.**ContextDataObject**(***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

Represents a context

PyOpenWorm.context_common module

PyOpenWorm.context_store module

exception PyOpenWorm.context_store.**ContextStoreException**
Bases: exceptions.Exception

PyOpenWorm.contextualize module

class PyOpenWorm.contextualize.**Contextualizable**(**args*, ***kwargs*)
Bases: PyOpenWorm.contextualize.BaseContextualizable

A BaseContextualizable with the addition of a default behavior of setting the context from the class's 'context' attribute. This generally requires that for the metaclass of the Contextualizable that a 'context' data property is defined. For example:

```
>>> class AMeta(ContextualizableClass):
>>>     @property
>>>     def context(self):
>>>         return self.__context
```

```
>>>     @context.setter
>>>     def context(self, ctx):
>>>         self.__context = ctx
```

```
>>> class A(six.with_metaclass(Contextualizable)):
>>>     pass
```

class PyOpenWorm.contextualize.**ContextualizableClass**
Bases: type

A super-type for contextualizable classes

PyOpenWorm.contextualize.**contextualize_helper**(*context*, *obj*, *noneok=False*)
Does some extra stuff to make access to the type of a ContextualizingProxy work more-or-less like access to the the wrapped object

PyOpenWorm.contextualize.**decontextualize_helper**(*obj*)
Removes contexts from a ContextualizingProxy

PyOpenWorm.data module

class PyOpenWorm.data.**Data**(*conf=None*, ***kwargs*)
Bases: *PyOpenWorm.configure.Configure*

Provides configuration for access to the database.

Usually doesn't need to be accessed directly

closeDatabase()
Close a the configured database

init_database()
Open the configured database

classmethod load(*file_name*)
Load a file into a new Data instance storing configuration in a JSON format

classmethod open(*file_name*)
Load a file into a new Data instance storing configuration in a JSON format

class PyOpenWorm.data.**DataUser**(**args*, ***kwargs*)
Bases: *PyOpenWorm.configure.Configureable*

A convenience wrapper for users of the database

Classes which use the database should inherit from DataUser.

add_reference(*g*, *reference_iri*)
Add a citation to a set of statements in the database

Parameters **triples** – A set of triples to annotate

add_statements(*graph*)
Add a set of statements to the database. Annotates the addition with uploader name, etc

Parameters **graph** – An iterable of triples

infer()
Fire FuXi rule engine to infer triples

retract_statements(*graph*)
Remove a set of statements from the database.

Parameters **graph** – An iterable of triples

class PyOpenWorm.data.**RDFSource**(***kwargs*)
Bases: *PyOpenWorm.configure.Configureable*, *PyOpenWorm.configure.ConfigValue*

Base class for data sources.

Alternative sources should dervie from this class

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

class PyOpenWorm.data.**SerializationSource**(***kwargs*)
Bases: *PyOpenWorm.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" = <rdflib serialization format>
"rdf.store_conf" = <your rdflib store configuration here>
```

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

class PyOpenWorm.data.**TrixSource**(***kwargs*)
Bases: *PyOpenWorm.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 PyOpenWorm.data.**SPARQLSource**(***kwargs*)
Bases: *PyOpenWorm.data.RDFSource*

Reads from and queries against a remote data store

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

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

class PyOpenWorm.data.**SleepyCatSource**(***kwargs*)
Bases: *PyOpenWorm.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>
```

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

class PyOpenWorm.data.**DefaultSource**(***kwargs*)
Bases: *PyOpenWorm.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.

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

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

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

PyOpenWorm.dataObject module

class PyOpenWorm.dataObject.**BaseDataObject**(***kwargs*)
Bases: PyOpenWorm.identifier_mixin._IdMixin, yarom.graphObject.GraphObject, *PyOpenWorm.data.DataUser*, *PyOpenWorm.contextualize.Contextualizable*

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

classmethod DatatypeProperty(**args*, ***kwargs*)
Attach a, possibly new, property to this class that has a simple type (string,number,etc) for its values

Parameters

linkName [string] The name of this property.

owner [PyOpenWorm.dataObject.BaseDataObject] The name of this property.

classmethod ObjectProperty(**args*, ***kwargs*)
Attach a, possibly new, property to this class that has a complex BaseDataObject for its values

Parameters

linkName [string] The name of this property.

owner [PyOpenWorm.dataObject.BaseDataObject] The name of this property.

value_type [type] The type of BaseDataObject for values of this property

classmethod UnionProperty(**args*, ***kwargs*)
Attach a, possibly new, property to this class that has a simple type (string,number,etc) or BaseDataObject for its values

Parameters

linkName [string] The name of this property.

owner [PyOpenWorm.dataObject.BaseDataObject] The name of this property.

clear_po_cache()
Clear the property-object cache for this object.

This cache is maintained by and shared by the properties of this object. It isn't necessary to clear this cache manually unless you modify the RDFLib graph indirectly (e.g., through the store) at runtime.

decontextualize()
Return the object with all contexts removed

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

graph_pattern(*shorten=False*, *show_namespaces=True*, ***kwargs*)
Get the graph pattern for this object.

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

Parameters

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

id_is_variable()
Is the identifier a variable?

retract()
Remove this object from the data store.

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

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.

po_cache = None
A cache of property URIs and values. Used by RealSimpleProperty

properties_are_init_args = True
If true, then properties defined in the class body can be passed as keyword arguments to __init__. For example:

```
>>> class A(DataObject):
...     p = DatatypeProperty()

>>> A(p=5)
```

If the arguments are written explicitly into the __init__, then no special processing is done.

class PyOpenWorm.dataObject.**ContextMappedClass**(*name*, *bases*, *dct*)
Bases: yarom.mappedClass.MappedClass, *PyOpenWorm.contextualize.ContextualizableClass*

after_mapper_module_load(*mapper*)
Called after all classes in a module have been loaded

definition_context
Unlike self.context, definition_context isn't meant to be overriden

class PyOpenWorm.dataObject.**DataObject**(***kwargs*)
Bases: *PyOpenWorm.dataObject.BaseDataObject*

class PyOpenWorm.dataObject.**values**(*group_name*, ***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

A convenience class for working with a collection of objects

Example:

```
v = values('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 = values('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

value [ObjectProperty] An object in the group

add [ObjectProperty] an alias for `value`

PyOpenWorm.dataObjectUtils module

PyOpenWorm.datasource module

exception PyOpenWorm.datasource.**DuplicateAlsoException**
Bases: exceptions.Exception

class PyOpenWorm.datasource.**BaseDataTranslator**
Bases: *PyOpenWorm.dataObject.BaseDataObject*

Translates from a data source to PyOpenWorm objects

input_type
alias of *DataSource*

output_type
alias of *DataSource*

translation_type
alias of *Translation*

get_data_objects(*data_source*)
Override this to change how data objects are generated

make_translation(*sources=()*)
It's intended that implementations of DataTranslator will override this method to make custom Translations according with how different arguments to Translate are (or are not) distinguished.

The actual properties of a Translation subclass must be defined within the 'translate' method

translate(**args*, ***kwargs*)
Notionally, this method takes a data source, which is translated into some other data source. There doesn't necessarily need to be an input data source.

class PyOpenWorm.datasource.**DataObjectContextDataSource**(*context*, ***kwargs*)
Bases: *PyOpenWorm.datasource.DataSource*

Description [DatatypeProperty] Attribute: *description*

Free-text describing the data source

Input source [ObjectProperty] Attribute: *source*

The data source that was translated into this one

Translation [ObjectProperty] Attribute: *translation*

Information about the translation process that created this object

class PyOpenWorm.datasource.**DataSource**(***kwargs*)
Bases: *PyOpenWorm.dataObject.BaseDataObject*

A source for data that can get translated into PyOpenWorm objects.

The value for any field can be passed to __init__ by name. Additionally, if the sub-class definition of a DataSource assigns a value for that field like:

```
class A(DataSource):
    some_field = 3
```

that value will be used over the default value for the field, but not over any value provided to __init__.

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.

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

class PyOpenWorm.datasource.**DataSourceType**(*name*, *bases*, *dct*)
Bases: *PyOpenWorm.dataObject.ContextMappedClass*

A type for DataSources

Sets up the graph with things needed for MappedClasses

class PyOpenWorm.datasource.**DataTransatorType**(*name*, *bases*, *dct*)
Bases: *PyOpenWorm.dataObject.ContextMappedClass*

class PyOpenWorm.datasource.**DataTranslator**
Bases: *PyOpenWorm.datasource.BaseDataTranslator*

A specialization with the GenericTranslation translation type that adds sources for the translation automatically when a new output is made

translation_type
alias of *GenericTranslation*

make_translation(*sources=()*)
It's intended that implementations of DataTranslator will override this method to make custom Translations according with how different arguments to Translate are (or are not) distinguished.

The actual properties of a Translation subclass must be defined within the 'translate' method

class PyOpenWorm.datasource.**GenericTranslation**(*source=None*, ***kwargs*)
Bases: *PyOpenWorm.datasource.Translation*

A generic translation that just has sources in order

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.

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

class PyOpenWorm.datasource.**PersonDataTranslator**(*person*)
Bases: *PyOpenWorm.datasource.BaseDataTranslator*

A person who was responsible for carrying out the translation of a data source

class PyOpenWorm.datasource.**Translation**(*translator=None*, ***kwargs*)
Bases: *PyOpenWorm.dataObject.BaseDataObject*

Representation of the method by which a DataSource was translated and the sources of that translation. Unlike the 'source' field attached to DataSources, the Translation may distinguish different kinds of input source to a translation.

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.

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

PyOpenWorm.document module

exception PyOpenWorm.document.**PubmedRetrievalException**
Bases: exceptions.Exception

exception PyOpenWorm.document.**WormbaseRetrievalException**
Bases: exceptions.Exception

class PyOpenWorm.document.**BaseDocument**(***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

class PyOpenWorm.document.**Document**(*bibtex=None*, *doi=None*, *pubmed=None*, *wormbase=None*, ***kwargs*)
Bases: *PyOpenWorm.document.BaseDocument*

A representation of some document.

Possible keys include:

```
pmid, pubmed: a pubmed id or url (e.g., 24098140)
wbid, wormbase: a wormbase id or url (e.g., WBPaper00044287)
doi: a Digitial Object id or url (e.g., s00454-010-9273-0)
uri: a URI specific to the document, preferably usable for accessing
     the document
```

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.

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

update_from_wormbase(*replace_existing=False*)
Queries wormbase for additional data to fill in the Document.

If replace_existing is set to *True*, then existing values will be cleared.

author
An author of the document

date
Alias to year

doi
A Digital Object Identifier (DOI), optional

pmid
A PubMed ID (PMID) that points to a paper

title
The title of the document

uri
A non-standard URI for the document

wbid
An ID from WormBase.org that points to a record, optional

wormbaseid
An alias to *wbid*

year
The year (e.g., publication year) of the document

PyOpenWorm.documentContext module

class PyOpenWorm.documentContext.**DocumentContext**(*document*)
Bases: *PyOpenWorm.context.Context*

A Context that corresponds to a document.

class PyOpenWorm.documentContext.**DocumentContextMeta**
Bases: *PyOpenWorm.context.ContextMeta*

PyOpenWorm.evidence module

exception PyOpenWorm.evidence.**EvidenceError**
Bases: exceptions.Exception

class PyOpenWorm.evidence.**Evidence**(***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

A representation which provides evidence, for a group of statements.

Attaching evidence to an set of statements is done like this:

```
>>> from PyOpenWorm.connection import Connection
>>> from PyOpenWorm.evidence import Evidence
>>> from PyOpenWorm.context import Context
```

Declare contexts:

```
>>> ACTX = Context(ident="http://example.org/data/some_statements")
>>> BCTX = Context(ident="http://example.org/data/some_other_statements")
>>> EVCTX = Context(ident="http://example.org/data/some_statements#evidence")
```

Make statements in *ACTX* and *BCTX* contexts:

```
>>> ACTX(Connection)(pre_cell="VA11", post_cell="VD12", number=3)
>>> BCTX(Connection)(pre_cell="VA11", post_cell="VD12", number=2)
```

In *EVCTX*, state that a that a certain document supports the set of statements in *ACTX*, but refutes the set of statements in *BCTX*:

```
>>> doc = EVCTX(Document)(author='White et al.', date='1986')
>>> EVCTX(Evidence)(reference=doc, supports=ACTX.rdf_object)
>>> EVCTX(Evidence)(reference=doc, refutes=BCTX.rdf_object)
```

Finally, save the contexts:

```
>>> ACTX.save_context()
>>> BCTX.save_context()
>>> EVCTX.save_context()
```

One note about the *reference* predicate: the reference should, ideally, be an unambiguous link to a peer-reviewed piece of scientific literature detailing methods and data analysis that supports the set of statements. However, in gather data from pre-existing sources, going to that level of specificity may be difficult due to deficient query capability at the data source. In such cases, a broader reference, such as a *Website* with information which guides readers to a peer-reviewed article supporting the statement is sufficient.

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

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

reference
The resource providing evidence supporting/refuting the attached context

refutes
A context naming a set of statements which are refuted by the attached reference

supports
A context naming a set of statements which are supported by the attached reference

PyOpenWorm.experiment module

class PyOpenWorm.experiment.**Experiment**(*reference=None*, ***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

Generic class for storing information about experiments

Should be overridden by specific types of experiments (example: see PatchClampExperiment in channelworm.py).

Overriding classes should have a list called "conditions" that contains the names of experimental conditions for that particular type of experiment. Each of the items in "conditions" should also be either a DatatypeProperty or ObjectProperty for the experiment as well.

Parameters

reference [Evidence] Supporting article for this experiment.

get_conditions()
Return conditions and their associated values in a dict.

PyOpenWorm.identifier_mixin module

PyOpenWorm.identifier_mixin.**IdMixin**(*typ=<type 'object'>*, *hashfunc=None*)

Parameters

typ [type] The type of object to use as the hash function's super class. Defaults to 'object'

hashfunc [function] The function to use for encoding data provided to make_identifier. Should return an object can `.encode()` to a `bytes` (a.k.a. `str` in Python 2). Defaults to `hashlib.sha224()`

PyOpenWorm.import_contextualizer module

class PyOpenWorm.import_contextualizer.**ImportContextualizer**

Bases: `object`

Interface for classes that 'contextualize' an import.

Contextualizing an import means that if an object is defined with the name X in some context, and an import statement is written like this:

```
import X.a
```

X will be invoked like this:

```
a = X(__import__('a'))
```

On the other hand, for an import statement of this form:

```
from X.a import A
```

will cause X to be invoked as:

```
_temp = X(__import__('a', globals(), locals(), ('A',)), ('A',))
A = _temp.A
```

and the import statement:

```
from X.a import A as AA
```

will cause X to be invoked as:

```
_temp = X(__import__('a', globals(), locals(), ('A',)), ('A',))
AA = _temp.A
```

meaning that the contextualizer won't know what the 'as' name is.

For the astute reader, you may notice parallels between the protocol for ImportContextualizer and the __import__ function itself. You may also be wondering why the contextualizer isn't merely a proxy for __import__. The first reason is that I want the true import to always happen, regardless of what the contextualizer does, so that the semantics of the contextualizer are very clear. The second reason is that requiring a real proxy would require more complexity in the contextualizers to ensure that they are properly handling exceptions, module attributes and the return value of __import__, ensuring that the *right* __import__ is used, as well as handling the 'fromlist' correctly.

PyOpenWorm.import_override module

PyOpenWorm.inverse_property module

For declaring inverse properties of GraphObjects

exception PyOpenWorm.inverse_property.**InversePropertyException**
Bases: exceptions.Exception

class PyOpenWorm.inverse_property.**InversePropertyMixin**
Bases: object

Mixin for inverse properties.

Augments RealSimpleProperty methods to update inverse properties as well

PyOpenWorm.muscle module

class PyOpenWorm.muscle.**Muscle**(*name=False*, ***kwargs*)
Bases: *PyOpenWorm.cell.Cell*

A single muscle cell.

See what neurons innervate a muscle:

Example:

```
>>> mdr21 = Muscle('MDR21')
>>> innervates_mdr21 = mdr21.innervatedBy()
>>> len(innervates_mdr21)
4
```

Attributes

neurons [ObjectProperty] Neurons synapsing with this muscle

receptors [DatatypeProperty] Get a list of receptors for this muscle if called with no arguments, or state that this muscle has the given receptor type if called with an argument

PyOpenWorm.my_neuroml module

class PyOpenWorm.my_neuroml.**NeuroML**(**args*, ***kwargs*)
Bases: *PyOpenWorm.data.DataUser*

classmethod generate(*o*, *t=2*)
Get a NeuroML object that represents the given object. The `type` determines what content is included in the NeuroML object:

Parameters

- **o** – The object to generate neuroml from
- **t** – The what kind of content should be included in the document - 0=full morphology+biophysics - 1=cell body only+biophysics - 2=full morphology only

Returns A NeuroML object that represents the given object.

Return type NeuroMLDocument

classmethod write(*o*, *n*)
Write the given neuroml document object out to a file :param o: The NeuroMLDocument to write :param n: The name of the file to write to

PyOpenWorm.network module

class PyOpenWorm.network.**Network**(*worm=None*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

A network of neurons

Attributes

neuron Returns a set of all Neuron objects in the network

synapse Returns a set of all synapses in the network

aneuron(*name*)
Get a neuron by name.

Example:

```
# Grabs the representation of the neuronal network
>>> net = Worm().get_neuron_network()

# Grab a specific neuron
>>> aval = net.aneuron('AVAL')

>>> aval.type()
set([u'interneuron'])
```

Parameters **name** – Name of a c. elegans neuron

Returns Neuron corresponding to the name given

Return type *PyOpenWorm.neuron.Neuron*

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.

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

interneurons()
Get all interneurons

Returns A iterable of all interneurons

Return type iter(*Neuron*)

motor()
Get all motor

Returns A iterable of all motor neurons

Return type iter(*Neuron*)

neuron_names()
Gets the complete set of neurons' names in this network.

Example:

```
# Grabs the representation of the neuronal network
>>> net = Worm().get_neuron_network()

#NOTE: This is a VERY slow operation right now
>>> len(set(net.neuron_names()))
302
>>> set(net.neuron_names())
set(['VB4', 'PDEL', 'HSNL', 'SIBDR', ... 'RIAL', 'MCR', 'LUAL'])
```

sensory()
Get all sensory neurons

Returns A iterable of all sensory neurons

Return type iter(*Neuron*)

PyOpenWorm.neuron module

class PyOpenWorm.neuron.**ConnectionProperty**(***kwargs*)
Bases: *PyOpenWorm.pProperty.Property*

A representation of the connection between neurons. Either a gap junction or a chemical synapse

TODO: Add neurotransmitter type. TODO: Add connection strength

get(*pre_post_or_either='pre'*, ***kwargs*)
Get a list of connections associated with the owning neuron.

Parameters

pre_post_or_either: str What kind of connection to look for. 'pre': Owner is the source of the connection 'post': Owner is the destination of the connection 'either': Owner is either the source or destination of the connection

Returns

list of Connection

set(*conn*, ***kwargs*)
Add a connection associated with the owner Neuron

Parameters

conn [PyOpenWorm.connection.Connection] connection associated with the owner neuron

Returns

A PyOpenWorm.neuron.Connection

class PyOpenWorm.neuron.**Neighbor**(***kwargs*)
Bases: *PyOpenWorm.pProperty.Property*

get(***kwargs*)
Get a list of neighboring neurons.

Parameters

See parameters for PyOpenWorm.connection.Connection

Returns

list of Neuron

set(*other*, ***kwargs*)
Set the value of this property

Derived classes must override.

class PyOpenWorm.neuron.**Neuron**(*name=False*, ***kwargs*)
Bases: *PyOpenWorm.cell.Cell*

A neuron.

See what neurons express some neuropeptide

Example:

```
# Grabs the representation of the neuronal network
>>> net = P.Worm().get_neuron_network()

# Grab a specific neuron
>>> aval = net.aneuron('AVAL')

>>> aval.type()
set([u'interneuron'])

#show how many connections go out of AVAL
>>> aval.connection.count('pre')
77

>>> aval.name()
u'AVAL'

#list all known receptors
>>> sorted(aval.receptors())
[u'GGR-3', u'GLR-1', u'GLR-2', u'GLR-4', u'GLR-5', u'NMR-1', u'NMR-2', u'UNC-8']

#show how many chemical synapses go in and out of AVAL
>>> aval.Syn_degree()
90
```

Parameters

name [string] The name of the neuron.

Attributes

type [DatatypeProperty] The neuron type (i.e., sensory, interneuron, motor)

receptor [DatatypeProperty] The receptor types associated with this neuron

innexin [DatatypeProperty] Innexin types associated with this neuron

neurotransmitter [DatatypeProperty] Neurotransmitters associated with this neuron

neuropeptide [DatatypeProperty] Name of the gene corresponding to the neuropeptide produced by this neuron

neighbor [Property] Get neurons connected to this neuron if called with no arguments, or with arguments, state that neuronName is a neighbor of this Neuron

connection [Property] Get a set of Connection objects describing chemical synapses or gap junctions between this neuron and others

GJ_degree()
Get the degree of this neuron for gap junction edges only

Returns total number of incoming and outgoing gap junctions

Return type int

Syn_degree()
Get the degree of this neuron for chemical synapse edges only

Returns total number of incoming and outgoing chemical synapses

Return type int

get_incidents(*type=0*)
Get neurons which synapse at this neuron

PyOpenWorm.pProperty module

class PyOpenWorm.pProperty.**Property**(*name=False*, *owner=False*, ***kwargs*)
Bases: *PyOpenWorm.contextualize.Contextualizable*, *PyOpenWorm.data.DataUser*

Store a value associated with a DataObject

Properties can be be accessed like methods. A method call like:

```
a.P()
```

for a property `P` will return values appropriate to that property for `a`, the *owner* of the property.

Parameters

owner [PyOpenWorm.dataObject.DataObject] The owner of this property

name [string] The name of this property. Can be accessed as an attribute like:

```
owner.name
```

get(**args*)
Get the things which are on the other side of this property

The return value must be iterable. For a `get` that just returns a single value, an easy way to make an iterable is to wrap the value in a tuple like `(value,)`.

Derived classes must override.

has_value()
Returns true if the Property has any values set on it.

This may be defined differently for each property

one()
Returns a single value for the `Property` whether or not it is multivalued.

set(**args*, ***kwargs*)
Set the value of this property

Derived classes must override.

class PyOpenWorm.pProperty.**PropertyMeta**(*name*, *bases*, *dct*)
Bases: *PyOpenWorm.contextualize.ContextualizableClass*

PyOpenWorm.package_utils module

PyOpenWorm.plot module

class PyOpenWorm.plot.**Plot**(*data=False*, **args*, ***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

Object for storing plot data in PyOpenWorm.

Parameters

data [2D list (list of lists)] List of XY coordinates for this Plot.

Example usage ::

```
>>> pl = Plot([[1, 2], [3, 4]])
>>> pl.get_data()
# [[1, 2], [3, 4]]
```

get_data()
Get the data stored for this plot.

set_data(*data*)
Set the data attribute, which is user-facing, as well as the serialized _data_string attribute, which is used for db storage.

PyOpenWorm.relationship module

class PyOpenWorm.relationship.**Relationship**(*s=None*, *p=None*, *o=None*, ***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

A Relationship is typically related to a property and is an object that one points to for talking about the property relationship.

For SimpleProperty objects, this acts like a RDF Reified triple.

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

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

PyOpenWorm.relationshipProxy module

class PyOpenWorm.relationshipProxy.**Rel**
Bases: tuple

A container for a relationship-assignment

PyOpenWorm.simpleProperty module

class PyOpenWorm.simpleProperty.**ContextMappedPropertyClass**(**args*, ***kwargs*)
Bases: yarom.mappedProperty.MappedPropertyClass, *PyOpenWorm.contextualize.ContextualizableClass*

class PyOpenWorm.simpleProperty.**DatatypeProperty**(*resolver*, ***kwargs*)
Bases: yarom.propertyMixins.DatatypePropertyMixin, PyOpenWorm.simpleProperty.PropertyCountMixin, *PyOpenWorm.simpleProperty.RealSimpleProperty*

class PyOpenWorm.simpleProperty.**ObjectProperty**(**args*, ***kwargs*)
Bases: *PyOpenWorm.inverse_property.InversePropertyMixin*, PyOpenWorm.simpleProperty._ContextualizingPropertySetMixin, yarom.propertyMixins.ObjectPropertyMixin, PyOpenWorm.simpleProperty.PropertyCountMixin, *PyOpenWorm.simpleProperty.RealSimpleProperty*

class PyOpenWorm.simpleProperty.**POCache**
Bases: tuple

The predicate-object cache object

class PyOpenWorm.simpleProperty.**RealSimpleProperty**(*owner*, ***kwargs*)
Bases: *PyOpenWorm.data.DataUser*, *PyOpenWorm.contextualize.Contextualizable*

clear()
Clears values set *in all contexts*

class PyOpenWorm.simpleProperty.**UnionProperty**(*resolver*, ***kwargs*)
Bases: PyOpenWorm.simpleProperty._ContextualizingPropertySetMixin, *PyOpenWorm.inverse_property.InversePropertyMixin*, yarom.propertyMixins.UnionPropertyMixin, PyOpenWorm.simpleProperty.PropertyCountMixin, *PyOpenWorm.simpleProperty.RealSimpleProperty*

A Property that can handle either DataObjects or basic types

PyOpenWorm.statement module

class PyOpenWorm.statement.**Statement**
Bases: *PyOpenWorm.statement.Statement*

PyOpenWorm.utils module

Common utilities for translation, massaging data, etc., that don't fit elsewhere in PyOpenWorm

PyOpenWorm.utils.**grouper**(*iterable*, *n*, *fillvalue=None*)
Collect data into fixed-length chunks or blocks

PyOpenWorm.website module

class PyOpenWorm.website.**Website**(*url=None*, *title=None*, ***kwargs*)
Bases: *PyOpenWorm.document.BaseDocument*

A representation of website

Attributes

url [DatatypeProperty] A URL for the website

title [DatatypeProperty] The official name for the website

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.

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

PyOpenWorm.worm module

class PyOpenWorm.worm.**Worm**(*scientific_name=False*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

A representation of the whole worm.

All worms with the same name are considered to be the same object.

Attributes

neuron_network [ObjectProperty] The neuron network of the worm

muscle [ObjectProperty] Muscles of the worm

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_neuron_network()
Return the neuron network of the worm.

Example:

```
# Grabs the representation of the neuronal network
>>> net = P.Worm().get_neuron_network()

# Grab a specific neuron
>>> aval = net.aneuron('AVAL')

>>> aval.type()
set([u'interneuron'])
```

(continues on next page)

(continued from previous page)

```
#show how many connections go out of AVAL
>>> aval.connection.count('pre')
77
```

Returns An object to work with the network of the worm

Return type PyOpenWorm.Network

get_semantic_net()

Get the underlying semantic network as an RDFLib Graph

Returns A semantic network containing information about the worm

Return type rdflib.ConjunctiveGraph

identifier_augment(**args*, ***kwargs*)

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

muscles()

Get all Muscle objects attached to the Worm.

Example:

```
>>> muscles = P.Worm().muscles()
>>> len(muscles)
96
```

Returns A set of all muscles

Return type `set`

CHAPTER 2

For Users

2.1 PyOpenWorm Data Sources

The sources of data for PyOpenWorm are stored under the OpenWormData/aux_data directory.

These data sources are brought into PyOpenWorm by means of the insert_worm.py script, which shows exactly how a given kind of data is brought into the system. For example this method shows where data about muscles comes from, while this method shows where the connectivity data comes from. A more detailed human readable key is provided below.

2.1.1 A Note on PyOpenWorm Data

Below, each major element of the worm's anatomy that PyOpenWorm stores data on is considered individually. The data being used is tagged by source in a superscript, and the decisions made during the curation process (if any) are described.

2.1.2 Neurons

- Neuron names[2]: Extracted from WormBase. Dynamic version on this google spreadsheet. Staged in this csv file. Parsed by this method.
- Neuron types[1]: Extracted from WormAtlas.org. Staged in this csv file. Parsed by this method.
- Cell descriptions[1]: Extracted from WormAtlas.org. Staged in this tsv file. Parsed by this method.

[2]
- Harris, T. W., Antoshechkin, I., Bieri, T., Blasiar, D., Chan, J., Chen, W. J., ... Sternberg, P. W. (2010). WormBase: a comprehensive resource for nematode research. Nucleic Acids Research, 38(Database issue), D463–7. http://doi.org/10.1093/nar/gkp952
- Lee, R. Y. N., & Sternberg, P. W. (2003). Building a cell and anatomy ontology of Caenorhabditis elegans. Comparative and Functional Genomics, 4(1), 121–6. http://doi.org/10.1002/cfg.248

[1] Altun, Z.F., Herndon, L.A., Wolkow, C.A., Crocker, C., Lints, R. and Hall, D. H. (2015). WormAtlas. Retrieved from http://www.wormatlas.org - WormAtlas Complete Cell List

- Lineage names[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this tsv file. Parsed by this method.
- Neurotransmitters[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this csv file. Parsed by this method.
- Neuropeptides[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this csv file. Parsed by this method.
- Receptors[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this csv file. Parsed by this method.
- Innexins[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this csv file. Parsed by this method.

Gene expression data below, additional to that extracted from WormAtlas concerning receptors, neuropeptides, neurotransmitters and innexins are parsed by this method:

- Monoamine secretors and receptors, neuropeptide secretors and receptors[4]: Dynamic version on this google spreadsheet. Staged in this csv file.

2.1.3 Muscle cells

- Muscle names[2]: Extracted from WormBase. Dynamic version on this google spreadsheet. Staged in this csv file. Parsed by this method.
- Cell descriptions[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this tsv file. Parsed by this method.
- Lineage names[1]: Extracted from WormAtlas.org. Dynamic version on this google spreadsheet. Staged in this tsv file. Parsed by this method.
- Neurons that innervate each muscle[3]: Extracted from data personally communicated by S. Cook. Staged in this csv file. Parsed by this method.

2.1.4 Connectome

- Gap junctions between neurons[3]: Extracted from data personally communicated by S. Cook. Staged in this csv file. Parsed by this method.
- Synapses between neurons[3]: Extracted from data personally communicated by S. Cook. Staged in this csv file. Parsed by this method.

Curation note

There was another source of C. *elegans* connectome data that was created by members of the OpenWorm project that has since been retired. The history of this spreadsheet is mostly contained in this forum post We decided to use the Emmons data set[3] as the authoritative source for connectome data, as it is the very latest version and updated version of the C. *elegans* connectome that we are familiar with.

---

[4] Bentley B., Branicky R., Barnes C. L., Chew Y. L., Yemini E., Bullmore E. T., Vertes P. E., Schafer W. R. (2016) The Multilayer Connectome of Caenorhabditis elegans. PLoS Comput Biol 12(12): e1005283. http://doi.org/10.1371/journal.pcbi.1005283

[3] Emmons, S., Cook, S., Jarrell, T., Wang, Y., Yakolev, M., Nguyen, K., Hall, D. Whole-animal C. elegans connectomes. C. Elegans Meeting 2015 http://abstracts.genetics-gsa.org/cgi-bin/celegans15s/wsrch15.pl?author=emmons&sort=ptimes&sbutton=Detail&absno=155110844&sid=668862

2.1.5 Data Source References

2.2 Requirements for data storage in OpenWorm

Our OpenWorm database captures facts about C. *elegans*. The database stores data for generating model files and together with annotations describing the origins of the data. Below are a set of recommendations for implementation of the database organized around an RDF model.

2.2.1 Interface

Access is through a Python library which communicates with the database. This library serves the function of providing an object oriented view on the database that can be accessed through the Python scripts commonly used in the project. The *api* is described separately.

2.2.2 Data modelling

Biophysical and anatomical data are included in the database. A sketch of some features of the data model is below. Also included in our model are the relationships between these types. Given our choice of data types, we do not model the individual interactions between cells as entities in the database. Rather these are described by generic predicates in an RDF triple. For instance, neuron A synapsing with muscle cell B would give a statement (A, synapsesWith, B), but A synapsing with neuron C would also have (A, synapsesWith, C). Data which belong to the specific relationship between two nodes is attached to an rdf:Statement object which points to the statement. This choice is intended to easy querying and extension later on.

Nervous system

In the worm's nervous system, we capture a few important data types (listed *below*). These correspond primarily to the anatomical structures and chemicals which are necessary for the worm to record external and internal stimuli and activate its body in response to those stimuli.

Data types

A non-exhaustive list of neurological data types in our C. elegans database:

- receptor types identified in the nerve cell
- neurons
- ion channels
- neurotransmitters
- muscle receptors

Development

Caenorhabditis elegans has very stable cell division patterns in the absence of mutations. This means that we can capture divisions in our database as static 'daughter_of' relationships. The theory of differentiation codes additionally gives an algorithmic description to the growth patterns of the worm which describes signals transmitted between developing cells. In order to test this theory we would like to leverage existing photographic data indicating the volume of cells at the time of their division as this relates to the differentiation code stored by the cell. Progress on this issue is documented on Github.

Aging

Concurrently with development, we would like to begin modeling the effects of aging on the worm. Aging typically manifests in physiological changes due to transcription errors or cell death. These physiological changes can be represented abstractly as parameters to the function of biological entities. See Github for further discussion.

2.2.3 Information assurance

Reasoning and Data integrity

To make full use of RDF storage it's recommended to leverage reasoning over our stored data. Encoding rules for the worm requires a good knowledge of both C. elegans and the database schema. More research needs to be done on this going forward. Preliminarily, SPIN, a constraint notation system based on SPARQL looks like a good candidate for *specifying* rules, but an inference engine for *enforcing* the rules still needs to be found.

Input validation

Input validation is to be handled through the interface library referenced *above*. In general, incorrect entry of biological names will result in an error being reported identifying the offending entry and providing a acceptable entries where appropriate. No direct access to the underlying data store will be provided.

Provenance

Tracking the origins of facts stated in the database demands a method of annotating statements in our database. Providing citations for facts must be as simple as providing a global identifier (e.g., URI, DOI) or a local identifier (e.g., Bibtex identifier, Pubmed ID). A technique called RDF reification allows us to annotate arbitrary facts in our database with additional information. This technique allows for the addition of structured citation data to facts in the database as well as annotations for tracking responsibility for uploads to the database. Further details for the attachment of evidence using this technique are given in the *api*.

In line with current practices for communication through the source code management platform, Github, we would like to track responsibility for new uploads to the database. Two methods are proposed for tracking this information: RDF named graphs and RDF reification. Tracking information must include, at least, a time-stamp on the update and linking of the submitted data to the uploader's unique identifier (e.g., email address). Named graphs have the advantage of wide support for the use of tracking uploads. The choice between these depends largely the support of the chosen data store for named graphs.

Access control

Write access to data in the project has been inconsistent between various data sources in the project. Going forward, write access to OpenWorm databases should be restricted to authenticated users to forestall the possibility of malicious tampering.

One way to accomplish this would be to leverage GitHub's fork and pull model with the data as well as the code. This would require two things:

- Instead of remote hosting of data, data is local to each copy of the library within a local database
- A serialization method dumps a new copy of the data out to a flat file enabling all users of the library to contribute their modifications to the data back to the PyOpenWorm project via GitHub.

A follow on to #2 is that the serialization method would need to preserve the ordering of data elements and write in some plain text format so that a simple diff on GitHub would be able to illuminate changes that were made.

2.2.4 Miscellaneous

Versioning

Experimental methods are constantly improving in biological research. These improvements may require updating the data we reference or store internally. However, in making updates we must not immediately expunge older content, breaking links created by internal and external agents. Ideally we would have a means of deprecating old data and specifying replacements. On the level of single resources, this is a trivial mapping which may be done transparently to all readers. For a more significant change, altering the schema, human intervention may be required to update external readers.

2.2.5 Why RDF?

RDF offers advantages in resilience to schema additions and increased flexibility in integrating data from disparate sources.[1] These qualities can be valued by comparison to relational database systems. Typically, schema changes in a relational database require extensive work for applications using it.[2] In the author's experience, RDF databases offer more freedom in restructuring. Also, for data integration, SPARQL, the standard language for querying over RDF has Federated queries which allow for nearly painless integration of external SPARQL endpoints with existing queries.

FuXi

FuXi is implemented as a semantic reasoning layer in PyOpenWorm. In other words, it will be used to automatically infer (and set) properties from other properties in the worm database. This means that redundant information (ex: explicitly stating that each object is of class "dataType") and subclass relationships (ex: that every object of type "Neuron" is also of type "Cell"), as well as other relationships, can be generated by the firing of FuXi's rule engine, without being hand-coded.

Aside from the time it saves in coding, FuXi may allow for a smaller footprint in the cloud, as many relationships within the database could be inferred *after* download.

A rule might be:

- { x is "Neuron" } => { x is "Cell" }

And a fact might be:

- { "ADLR" is "Neuron" }

Given the above rule and fact, FuXi could infer the new fact:

- { "ADLR" is "Cell" }

The advantage of local storage of the database that goes along with each copy of the library is that the data will have the version number of the library. This means that data can be 'deprecated' along with a deprecated version of the library. This also will prevent changes made to a volatile database that break downstream code that uses the library.

2.3 Adding Data to *YOUR* OpenWorm Database

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

You've come to the right place!

[1] http://answers.semanticweb.com/questions/19183/advantages-of-rdf-over-relational-databases
[2] http://research.microsoft.com/pubs/118211/andy%20maule%20-%20thesis.pdf

A few biological entities (e.g., Cell, Neuron, Muscle, Worm) are pre-coded into PyOpenWorm. The full list is available in the *API*. If these entities already cover your use-case, then all you need to do is add values for the appropriate fields and save them. If you have data already loaded into your database, then you can load objects from it:

```
n = Neuron()
n.receptor('UNC-13')
for x in n.load():
    do_something_with_unc13_neuron(n)
```

If you need additional entities it's easy to create them. Documentation for this is provided *here*.

Typically, you'll want to attach the data that you insert to entities already in the database. This allows you to recover objects in a hierarchical fashion from the database later. `Worm`, for instance has a property, `neuron_network`, which points to the `Network` which should contain all neural cells and synaptic connections. To initialize the hierarchy you would do something like:

```
ctx = Context(ident='http://example.org/c-briggsae')
w = ctx(Worm)('C. briggsae') # The name is optional and currently defaults to 'C.␣
↪elegans'
nn = ctx(Network)()          # make a neuron network
w.neuron_network(nn)         # attach to the worm the neuron network
n = ctx(Neuron)()            # make an unnamed neuron
n.receptor('UNC-13')         # state that the neuron has a UNC-13 type receptor
nn.neuron(n)                 # attach to the neuron network
ctx.save_context()           # save all of the data attached to the worm
```

It is possible to create objects without attaching them to anything and they can still be referenced by calling load on an instance of the object's class as in `n.load()` above. This also points out another fact: you don't have to set up the hierarchy for each insert in order for the objects to be linked to existing entities. If you have previously set up connections to an entity (e.g., `Worm('C. briggsae')`), assuming you *only* have one such entity, you can refer to things attached to it without respecifying the hierarchy for each script. The database packaged with PyOpenWorm should have only one Worm and one Network.

Remember that once you've set up all of the data, you must save the objects. For now, this requires keeping track of top-level objects – objects which aren't values of some other property – and calling `save()` on each of them individually. This isn't too difficult to achieve.

Future capabilities:

- Adding propositional logic to support making statements about all entities matching some conditions without needing to `load()` and `save()` them from the database.
- Statements like:

  ```
  ctx = Context(ident='http://example.org/c-briggsae')
  w = ctx.stored(Worm)()
  w.neuron_network.neuron.receptor('UNC-13')
  l = list(w.load()) # Get a list of worms with neurons expressing 'UNC-13'
  ```

  currently, to do the equivalent, you must work backwards, finding all neurons with UNC-13 receptors, then getting all networks with those neurons, then getting all worms with those networks:

  ```
  worms = set()
  n = ctx.stored(Neuron)()
  n.receptor('UNC-13')
  for ns in n.load():
      nn = ctx.stored(Network)()
      nn.neuron(ns)
  ```

(continues on next page)

(continued from previous page)

```
    for z in nn.load():
        w = ctx.stored(Worm)()
        w.neuron_network(z)
        worms.add(w)
l = list(worms)
```

It's not difficult logic, but it's 8 extra lines of code for a, conceptually, very simple query.

- Also, queries like:

```
l = list(ctx.stored(Worm)('C. briggsae').neuron_network.neuron.receptor()) # get␣
↪a list
#of all receptors expressed in neurons of C. briggsae
```

Again, not difficult to write out, but in this case it actually gives a much longer query time because additional values are queried in a `load()` call that are never returned.

We'd also like operators for composing many such strings so:

```
ctx.stored(Worm)('C. briggsae').neuron_network.neuron.get('receptor', 'innexin')
↪# list
#of (receptor, innexin) values for each neuron
```

would be possible with one query and thus not requiring parsing and iterating over neurons twice–it's all done in a single, simple query.

2.3.1 Contexts

Above, we didn't qualify our statements in any way, but stated them as simple facts. In reality, our statements are made in a context that influences how they should be interpreted. In PyOpenWorm, that context-sensitivity is modeled by using :class:`Context` objects. To see what this looks like, let's start with an example.

Example 1

I have data about widgets from BigDataWarehouse (BDW) that I want to translate into RDF using PyOpenWorm, but I don't want put them with my other widget data since BDW data may conflict with mine. Also, if get more BDW data, I want to be able to relate these data to that. A good way to keep data which are made at distinct times or which come from different, possibly conflicting, sources is using contexts. The code below shows how to do that:

```
from rdflib import ConjunctiveGraph
from PyOpenWorm.context import Context
from mymod import Widget # my model for Widgets
from bdw import Load # BigDataWarehouse API

# Create a Context with an identifier appropriate to this BDW data import
ctx = Context(ident='http://example.org/data/imports/BDW_Widgets_2017-2018')

# Create a context manager using the default behavior of reading the
# dictionary of current local variables
with ctx(W=Widget) as c:
    for record in Load(data_set='Widgets2017-2018'):
        # declares Widgets in this context
        c.W(part_number=record.pnum,
            fullness=record.flns,
```

(continues on next page)

(continued from previous page)

```
            hardiness=record.hrds)

# Create an RDFLib graph as the target for the data
g = ConjunctiveGraph()

# Save the data
c.save_context(g)

# Serialize the data in the nquads format so we can see that all of our
# statements are in the proper context
print(g.serialize(format='nquads'))
```

If you've worked with lots of data before, this kind of pattern should be familiar. You can see how, with later imports, you would follow the naming scheme to create new contexts (e.g., `http://example.org/data/imports/BDW_Widgets_2018-2019`).

2.4 Making data objects

To make new objects like `Neuron` or `Worm`, for the most part, you just need to make a Python class. Say, for example, that I want to record some information about drug reactions in C. elegans. I make `Drug` and `Experiment` classes to describe C. elegans reactions:

```
from PyOpenWorm import (DataObject,
                        DatatypeProperty,
                        ObjectProperty,
                        Worm,
                        Evidence,
                        connect)

class Drug(DataObject):
    # We set up properties in __init__
    def __init__(self,drug_name=False,*args,**kwargs):
        # pass arguments to DataObject
        DataObject.__init__(self,*args,**kwargs)
        Drug.DatatypeProperty('name', owner=self)
        if drug_name:
            self.name(drug_name)

class Experiment(DataObject):
    def __init__(self,*args,**kwargs):
        # pass arguments to DataObject
        DataObject.__init__(self,*args,**kwargs)
        Experiment.ObjectProperty('drug', value_type=Drug, owner=self)
        Experiment.ObjectProperty('subject', value_type=Worm, owner=self)
        Experiment.DatatypeProperty('route_of_entry', owner=self)
        Experiment.DatatypeProperty('reaction', owner=self)

connect()
# Set up with the RDF translation machinery
Experiment.register()
Drug.register()
```

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

```
d = Drug('moon rocks')
e = Experiment()
w = Worm("C. elegans")
ev = Evidence(author="Aperture Labs")
e.subject(w)
e.drug(d)
e.route_of_entry('ingestion')
e.reaction('no reaction')
ev.asserts(e)
```

and save it:

```
ev.save()
```

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

You can also add properties to an object after it has been created by calling either ObjectProperty or DatatypeProperty on the object as is done in `__init__`:

```
d = Drug('moon rocks')
Drug.DatatypeProperty('granularity', owner=self)
d.granularity('ground up')
```

Properties added in this fashion will not propagate to any other objects, but they will be saved along with the object they are attached to.

2.5 Sharing Data with other users

Sharing is key to PyOpenWorm. This document covers the appropriate way to share changes with other PyOpenWorm users.

The shared PyOpenWorm database is stored in a Git repository distinct from the PyOpenWorm source code. Currently the database is stored in a Github repository here .

When you create a database normally, it will be stored in a format which is opaque to humans. In order to share your database you have two options: You can share the scripts which are used to create your database or you can share a human-readable serialization of the database. The second option is better since it doesn't require re-running your script to use the generated data, but it is best to share both.

For sharing the serialization, you should first clone the repository linked above, read the current serialization into your database (see *below* for an example of how you would do this), and then write out the serialization:

```
import PyOpenWorm as P
P.connect('path/to/your/config/file')
P.config()['rdf.graph'].serialize('out.n3', format='n3')
P.disconnect()
```

Commit, your changes to the git repository, push to a fork of the repository on Github and submit a pull request on the main repository. If for some reason your are unwilling or unable to create a Github account, post to the OpenWorm-discuss mailing list with a patch on the main repository with your changes and someone will have a look, possibly ask for adjustments or justification for your addition, and ultimately merge the changes for you.

To read the database back in you would do something like:

```
import PyOpenWorm as P
P.connect('path/to/your/config/file')
P.config()['rdf.graph'].parse('out.n3', format='n3')
P.disconnect()
```

Scripts are also added to the repository on Github to the `scripts` subdirectory.

CHAPTER 3

For Developers

3.1 Testing in PyOpenWorm

3.1.1 Running tests

Tests should be run via setup.py like:

```
python setup.py test
```

you can pass options to `pytest` like so:

```
python setup.py test --addopts '-k DataIntegrityTest'
```

3.1.2 Writing tests

Tests are written using Python's unittest. In general, a collection of closely related tests should be in one file. For selecting different classes of tests, tests can also be tagged using pytest marks like:

```
@pytest.mark.tag
class TestClass(unittest.TestCase):
    ...
```

Currently, marks are used to distinguish between unit-level tests and others which have the `inttest` mark

3.2 Adding documentation

Documentation for PyOpenWorm is housed in two locations:

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

By way of example, 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.

3.2.1 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.

3.2.2 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

3.2.3 Conventions

If you'd like to add a convention, list it here and start using it. It can be reviewed as part of a pull request.

1. Narrative text should be wrapped at 80 characters.
2. Long links should be extracted from narrative text. Use your judgement on what 'long' is, but if it causes the line width to stray beyond 80 characters that's a good indication.

3.3 RDF semantics for PyOpenWorm

In the context of PyOpenWorm, biological objects are classes of, for instance, anatomical features of a worm. That is to say, statements made about C. elegans are not about a specific worm, but are stated about the entire class of worms. The semantics of a `property SimpleProperty/value value` triple are that if any value is set, then without any additional statements being made, an instance of the object has been observed to have the value at some point in time, somewhere, under some set of conditions. In other words, the statement is an existential quantification over the associated object(class).

The purpose of the identifiers for Properties is to allow statements to be made about them directly. An example:

```
<http://openworm.org/entities/Entity/1> <http://openworm.org/entities/Entity/
↪interactsWith> <http://openworm.org/entities/Entity_interactsWith/2> .
<http://openworm.org/entities/Entity_interactsWith/2> <http://openworm.org/entities/
↪SimpleProperty/value> <http://openworm.org/entities/Entity/3> .

<http://openworm.org/entities/Entity/4> <http://openworm.org/entities/Entity/
↪modulates> <http://openworm.org/entities/Entity_modulates/5> .
<http://openworm.org/entities/Entity_modulates/5> <http://openworm.org/entities/
↪SimpleProperty/value> <http://openworm.org/entities/Entity_interactsWith/2>
```

3.4 RDF structure for PyOpenWorm

For most use cases, it is (hopefully) not necessary to write custom queries over the RDF graph in order to work with PyOpenWorm. However, if it does become necessary, it will be helpful to have an understanding of the structure of the RDF graph. Thus, a summary is given below.

For all *DataObjects* which are not *Properties*, there is an identifier of the form

```
<http://openworm.org/entities/Object_type/md5sum>
```

stored in the graph. This identifier will be associated with type data:

```
<http://openworm.org/entities/Object_type/md5sum> rdf:type <http://openworm.org/
↪entities/Object_type> .
<http://openworm.org/entities/Object_type/md5sum> rdf:type <http://openworm.org/
↪entities/parent_of_Object_type> .
<http://openworm.org/entities/Object_type/md5sum> rdf:type <http://openworm.org/
↪entities/parent_of_parent_of_Object_type> .
...
```

Properties have a slightly different form. They also have an identifier, which for `SimpleProperties` will look like this:

```
<http://openworm.org/entities/OwnerType_propertyName/md5sum>
```

`OwnerType` is the type of the Property's owner and `propertyName` is the name by which the property is accessed from an object of the owner's type. Other Properties will not necessarily have this form, but all of the standard Properties are implemented in terms of SimpleProperties and have no direct representation in the graph. For other Properties it is necessary to refer to their documentation or to examine the triples released by the Property of interest.

A DataObject's identifier is connected to a property in a triple like:

```
<http://openworm.org/entities/OwnerType/md5sum> <http://openworm.org/entities/
↪OwnerType/propertyName> <http://openworm.org/entities/OwnerType_propertyName/md5sum>
```

and the property is connected to its values like:

```
<http://openworm.org/entities/OwnerType_propertyName/md5sum> <http://openworm.org/
↪entities/SimpleProperty/value> "A literal value"
```

The following API calls do not yet exist, but would be excellent next functions to implement

3.5 Population()

A collection of cells. Constructor creates an empty population.

3.5.1 Population.filterCells(filters : ListOf(PairOf(unboundMethod, methodArgument))) : Population

Allows for groups of cells to be created based on shared properties including neurotransmitter, anatomical location or region, cell type.

Example:

```
p = Worm.cells()
p1 = p.filterCells([(Cell.lineageName, "AB")]) # A population of cells with AB as the␣
↪blast cell
```

3.6 NeuroML()

A utility for generating NeuroML files from other objects. The semantics described *above* do not apply here.

3.6.1 NeuroML.generate(object : {Network, Neuron, IonChannel}, type : {0,1,2}) : neuroml.NeuroMLDocument

Get a NeuroML object that represents the given object. The `type` determines what content is included in the NeuroML object:

- 0=full morphology+biophysics
- 1=cell body only+biophysics
- 2=full morphology only

3.6.2 NeuroML.write(document : neuroml.NeuroMLDocument, filename : String)

Write out a NeuroMLDocument

3.7 PyOpenWorm coding standards

Pull requests are *required* to follow the PEP8 Guidelines for contributions of Python code to PyOpenWorm (with some exceptions). Compliance can be checked with the `pep8` tool and these command line arguments:

```
--max-line-length=120 --ignore=E261,E266,E265,E402,E121,E123,E126,E226,E24,E704
```

Some violations can be corrected with `autopep8`.

CHAPTER 4

Issues

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

PyOpenWorm, 3
PyOpenWorm.bibtex, 11
PyOpenWorm.biology, 12
PyOpenWorm.cell, 12
PyOpenWorm.channel, 13
PyOpenWorm.channelworm, 14
PyOpenWorm.command, 15
PyOpenWorm.configure, 16
PyOpenWorm.connection, 17
PyOpenWorm.context, 17
PyOpenWorm.context_common, 18
PyOpenWorm.context_store, 18
PyOpenWorm.contextDataObject, 18
PyOpenWorm.contextualize, 18
PyOpenWorm.data, 19
PyOpenWorm.data_trans, 4
PyOpenWorm.data_trans.common_data, 4
PyOpenWorm.data_trans.connections, 4
PyOpenWorm.data_trans.csv_ds, 6
PyOpenWorm.data_trans.data_with_evidence_ds, 6
PyOpenWorm.data_trans.local_file_ds, 7
PyOpenWorm.data_trans.neuron_data, 7
PyOpenWorm.data_trans.wormatlas, 8
PyOpenWorm.data_trans.wormbase, 9
PyOpenWorm.dataObject, 21
PyOpenWorm.dataObjectUtils, 23
PyOpenWorm.datasource, 23
PyOpenWorm.document, 26
PyOpenWorm.documentContext, 27
PyOpenWorm.evidence, 27
PyOpenWorm.experiment, 28
PyOpenWorm.identifier_mixin, 29
PyOpenWorm.import_contextualizer, 29
PyOpenWorm.import_override, 30
PyOpenWorm.inverse_property, 30
PyOpenWorm.muscle, 30
PyOpenWorm.my_neuroml, 30
PyOpenWorm.network, 31
PyOpenWorm.neuron, 32
PyOpenWorm.package_utils, 35
PyOpenWorm.plot, 35
PyOpenWorm.pProperty, 34
PyOpenWorm.relationship, 35
PyOpenWorm.relationshipProxy, 36
PyOpenWorm.simpleProperty, 36
PyOpenWorm.statement, 36
PyOpenWorm.utils, 36
PyOpenWorm.website, 37
PyOpenWorm.worm, 37

Index

A

add_graph() (PyOpenWorm.command.POW method), 15
add_reference() (PyOpenWorm.data.DataUser method), 19
add_statements() (PyOpenWorm.data.DataUser method), 19
after_mapper_module_load() (PyOpenWorm.dataObject.ContextMappedClass method), 22
aneuron() (PyOpenWorm.network.Network method), 31
author (PyOpenWorm.document.Document attribute), 26
author() (in module PyOpenWorm.bibtex), 12

B

BadConf, 16
BaseDataObject (class in PyOpenWorm.dataObject), 21
BaseDataTranslator (class in PyOpenWorm.datasource), 23
BaseDocument (class in PyOpenWorm.document), 26
bibtex_to_document() (in module PyOpenWorm.bibtex), 12
BibTexDataSource (class in PyOpenWorm.bibtex), 11
BibTexDataTranslator (class in PyOpenWorm.bibtex), 11
BiologyType (class in PyOpenWorm.biology), 12
blast() (PyOpenWorm.cell.Cell method), 12

C

Cell (class in PyOpenWorm.cell), 12
Channel (class in PyOpenWorm.channel), 13
ChannelModel (class in PyOpenWorm.channelworm), 14
clear() (PyOpenWorm.simpleProperty.RealSimpleProperty method), 36
clear_po_cache() (PyOpenWorm.dataObject.BaseDataObject method), 22
clone() (PyOpenWorm.command.POW method), 15
closeDatabase() (PyOpenWorm.data.Data method), 19
commit() (PyOpenWorm.command.POW method), 15
config() (in module PyOpenWorm), 4
config_file (PyOpenWorm.command.POW attribute), 16
Configure (class in PyOpenWorm.configure), 16
Configureable (class in PyOpenWorm.configure), 17
ConfigValue (class in PyOpenWorm.configure), 16
connect() (in module PyOpenWorm), 4
Connection (class in PyOpenWorm.connection), 17
ConnectionProperty (class in PyOpenWorm.neuron), 32
ConnectomeCSVDataSource (class in PyOpenWorm.data_trans.connections), 4
Context (class in PyOpenWorm.context), 17
ContextContextManager (class in PyOpenWorm.context), 18
ContextDataObject (class in PyOpenWorm.contextDataObject), 18
ContextMappedClass (class in PyOpenWorm.dataObject), 22
ContextMappedPropertyClass (class in PyOpenWorm.simpleProperty), 36
ContextMeta (class in PyOpenWorm.context), 18
ContextStoreException, 18
Contextualizable (class in PyOpenWorm.contextualize), 18
ContextualizableClass (class in PyOpenWorm.contextualize), 18
contextualize_helper() (in module PyOpenWorm.contextualize), 18
copy() (PyOpenWorm.configure.Configure method), 16
CSVDataSource (class in PyOpenWorm.data_trans.csv_ds), 6
CSVDataTranslator (class in PyOpenWorm.data_trans.csv_ds), 6
customizations() (in module PyOpenWorm.bibtex), 12

D

Data (class in PyOpenWorm.data), 19
data_source_type (PyOpenWorm.bibtex.BibTexDataTranslator attribute), 12
DataObject (class in PyOpenWorm.dataObject), 23

DataObjectContextDataSource (class in PyOpenWorm.datasource), 24
DataSource (class in PyOpenWorm.datasource), 24
DataSourceType (class in PyOpenWorm.datasource), 24
DataTransatorType (class in PyOpenWorm.datasource), 25
DataTranslator (class in PyOpenWorm.datasource), 25
DatatypeProperty (class in PyOpenWorm.simpleProperty), 36
DatatypeProperty() (PyOpenWorm.dataObject.BaseDataObject class method), 21
DataUser (class in PyOpenWorm.data), 19
DataWithEvidenceDataSource (class in PyOpenWorm.data_trans.data_with_evidence_ds), 6
date (PyOpenWorm.document.Document attribute), 26
decontextualize() (PyOpenWorm.dataObject.BaseDataObject method), 22
decontextualize_helper() (in module PyOpenWorm.contextualize), 18
DefaultSource (class in PyOpenWorm.data), 20
defined_augment() (PyOpenWorm.cell.Cell method), 12
defined_augment() (PyOpenWorm.data_trans.connections.NeuronConnectomeCSVTranslation method), 5
defined_augment() (PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataTranslation method), 8
defined_augment() (PyOpenWorm.datasource.DataSource method), 24
defined_augment() (PyOpenWorm.datasource.GenericTranslation method), 25
defined_augment() (PyOpenWorm.datasource.Translation method), 25
defined_augment() (PyOpenWorm.document.Document method), 26
defined_augment() (PyOpenWorm.evidence.Evidence method), 28
defined_augment() (PyOpenWorm.network.Network method), 31
defined_augment() (PyOpenWorm.relationship.Relationship method), 35
defined_augment() (PyOpenWorm.website.Website method), 37
defined_augment() (PyOpenWorm.worm.Worm method), 37
definition_context (PyOpenWorm.dataObject.ContextMappedClass attribute), 23
description (PyOpenWorm.cell.Cell attribute), 13
diff() (PyOpenWorm.command.POW method), 15
disconnect() (in module PyOpenWorm), 4
divisionVolume (PyOpenWorm.cell.Cell attribute), 13
Document (class in PyOpenWorm.document), 26
DocumentContext (class in PyOpenWorm.documentContext), 27
DocumentContextMeta (class in PyOpenWorm.documentContext), 27
doi (PyOpenWorm.document.Document attribute), 26
doi() (in module PyOpenWorm.bibtex), 12
DuplicateAlsoException, 23

E

Evidence (class in PyOpenWorm.evidence), 27
EvidenceError, 27
Experiment (class in PyOpenWorm.experiment), 28
ExpressionPattern (class in PyOpenWorm.channel), 13

F

fetch_graph() (PyOpenWorm.command.POW method), 15

G

generate() (PyOpenWorm.my_neuroml.NeuroML class method), 30
GenericTranslation (class in PyOpenWorm.datasource), 25
get() (PyOpenWorm.configure.Configure method), 16
get() (PyOpenWorm.configure.Configureable method), 17
get() (PyOpenWorm.neuron.ConnectionProperty method), 32
get() (PyOpenWorm.neuron.Neighbor method), 32
get() (PyOpenWorm.pProperty.Property method), 34
get_conditions() (PyOpenWorm.experiment.Experiment method), 28
get_data() (PyOpenWorm.plot.Plot method), 35
get_data_objects() (PyOpenWorm.datasource.BaseDataTranslator method), 24
get_incidents() (PyOpenWorm.neuron.Neuron method), 34
get_neuron_network() (PyOpenWorm.worm.Worm method), 37
get_owners() (PyOpenWorm.dataObject.BaseDataObject method), 22
get_semantic_net() (PyOpenWorm.worm.Worm method), 38
GJ_degree() (PyOpenWorm.neuron.Neuron method), 34
graph_accessor_finder (PyOpenWorm.command.POW attribute), 16
graph_pattern() (PyOpenWorm.dataObject.BaseDataObject method), 22

grouper() (in module PyOpenWorm.utils), 36

H

has_value() (PyOpenWorm.pProperty.Property method), 34
HomologyChannelModel (class in PyOpenWorm.channelworm), 14

I

id_is_variable() (PyOpenWorm.dataObject.BaseDataObject method), 22
identifier_augment() (PyOpenWorm.cell.Cell method), 13
identifier_augment() (PyOpenWorm.data_trans.connections.NeuronConnectomeCSVTranslation method), 5
identifier_augment() (PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataTranslation method), 8
identifier_augment() (PyOpenWorm.datasource.DataSource method), 24
identifier_augment() (PyOpenWorm.datasource.GenericTranslation method), 25
identifier_augment() (PyOpenWorm.datasource.Translation method), 25
identifier_augment() (PyOpenWorm.document.Document method), 26
identifier_augment() (PyOpenWorm.evidence.Evidence method), 28
identifier_augment() (PyOpenWorm.network.Network method), 31
identifier_augment() (PyOpenWorm.relationship.Relationship method), 35
identifier_augment() (PyOpenWorm.website.Website method), 37
identifier_augment() (PyOpenWorm.worm.Worm method), 38
IdMixin() (in module PyOpenWorm.identifier_mixin), 29
ImportContextualizer (class in PyOpenWorm.import_contextualizer), 29
infer() (PyOpenWorm.data.DataUser method), 19
init() (PyOpenWorm.command.POW method), 15
init_database() (PyOpenWorm.data.Data method), 19
input_type (PyOpenWorm.data_trans.neuron_data.NeuronCSVDataTranslator attribute), 7
input_type (PyOpenWorm.data_trans.wormbase.MuscleWormBaseCSVTranslator attribute), 9
input_type (PyOpenWorm.data_trans.wormbase.NeuronWormBaseCSVTranslator attribute), 9
input_type (PyOpenWorm.data_trans.wormbase.WormbaseInCharCSVTranslator attribute), 10
input_type (PyOpenWorm.data_trans.wormbase.WormbaseTextMatchCSVT attribute), 11
input_type (PyOpenWorm.datasource.BaseDataTranslator attribute), 23
interneurons() (PyOpenWorm.network.Network method), 31
InvalidGraphException, 15
InversePropertyException, 30
InversePropertyMixin (class in PyOpenWorm.inverse_property), 30

L

lineageName (PyOpenWorm.cell.Cell attribute), 13
link() (PyOpenWorm.configure.Configure method), 17
load() (PyOpenWorm.data.Data class method), 19
loadConfig() (in module PyOpenWorm), 3
loadData() (in module PyOpenWorm), 4
LocalFileDataSource (class in PyOpenWorm.data_trans.local_file_ds), 7

M

make_translation() (PyOpenWorm.data_trans.connections.NeuronConnectomeCSVTranslator method), 5
make_translation() (PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataTranslator method), 9
make_translation() (PyOpenWorm.datasource.BaseDataTranslator method), 24
make_translation() (PyOpenWorm.datasource.DataTranslator method), 25
merge() (PyOpenWorm.command.POW method), 16
motor() (PyOpenWorm.network.Network method), 31
Muscle (class in PyOpenWorm.muscle), 30
muscles() (PyOpenWorm.worm.Worm method), 38
MuscleWormBaseCSVTranslator (class in PyOpenWorm.data_trans.wormbase), 9

N

name (PyOpenWorm.cell.Cell attribute), 13
Neighbor (class in PyOpenWorm.neuron), 32
Network (class in PyOpenWorm.network), 31
NeuroML (class in PyOpenWorm.my_neuroml), 30
Neuron (class in PyOpenWorm.neuron), 33
neurons() (PyOpenWorm.network.Network method), 32
NeuronConnectomeCSVTranslation (class in PyOpenWorm.data_trans.connections), 5
NeuronCSVTranslator (class in PyOpenWorm.data_trans.connections), 5
NeuronCSVDataTranslator (class in PyOpenWorm.data_trans.neuron_data), 7

NeuronCSVDataTranslator (class in PyOpenWorm.data_trans.neuron_data), 7
NeuronWormBaseCSVTranslator (class in PyOpenWorm.data_trans.wormbase), 9

O

ObjectProperty (class in PyOpenWorm.simpleProperty), 36
ObjectProperty() (PyOpenWorm.dataObject.BaseDataObject class method), 21
one() (PyOpenWorm.pProperty.Property method), 34
open() (PyOpenWorm.configure.Configure class method), 17
open() (PyOpenWorm.data.Data class method), 19
open() (PyOpenWorm.data.DefaultSource method), 21
open() (PyOpenWorm.data.RDFSource method), 19
open() (PyOpenWorm.data.SerializationSource method), 20
open() (PyOpenWorm.data.SleepyCatSource method), 20
open() (PyOpenWorm.data.SPARQLSource method), 20
open() (PyOpenWorm.data.ZODBSource method), 21
output_type (PyOpenWorm.data_trans.connections.NeuronConnectomeCSVTranslator attribute), 5
output_type (PyOpenWorm.data_trans.neuron_data.NeuronCSVDataTranslator attribute), 7
output_type (PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataTranslator attribute), 8
output_type (PyOpenWorm.data_trans.wormbase.MuscleWormBaseCSVTranslator attribute), 9
output_type (PyOpenWorm.data_trans.wormbase.NeuronWormBaseCSVTranslator attribute), 9
output_type (PyOpenWorm.data_trans.wormbase.WormbaseIonChannelCSVTranslator attribute), 10
output_type (PyOpenWorm.data_trans.wormbase.WormbaseTextMatchCSVTranslator attribute), 11
output_type (PyOpenWorm.datasource.BaseDataTranslator attribute), 23

P

PatchClampChannelModel (class in PyOpenWorm.channelworm), 14
PatchClampExperiment (class in PyOpenWorm.channelworm), 14
PersonDataTranslator (class in PyOpenWorm.datasource), 25
Plot (class in PyOpenWorm.plot), 35
pmid (PyOpenWorm.document.Document attribute), 26
po_cache (PyOpenWorm.dataObject.BaseDataObject attribute), 22
POCache (class in PyOpenWorm.simpleProperty), 36
POW (class in PyOpenWorm.command), 15
properties_are_init_args (PyOpenWorm.dataObject.BaseDataObject attribute), 22
Property (class in PyOpenWorm.pProperty), 34
PropertyMeta (class in PyOpenWorm.pProperty), 35
PubmedRetrievalException, 26
push() (PyOpenWorm.command.POW method), 16
PyOpenWorm (module), 3
PyOpenWorm.bibtex (module), 11
PyOpenWorm.biology (module), 12
PyOpenWorm.cell (module), 12
PyOpenWorm.channel (module), 13
PyOpenWorm.channelworm (module), 14
PyOpenWorm.command (module), 15
PyOpenWorm.configure (module), 16
PyOpenWorm.connection (module), 17
PyOpenWorm.context (module), 17
PyOpenWorm.context_common (module), 18
PyOpenWorm.context_store (module), 18
PyOpenWorm.contextDataObject (module), 18
PyOpenWorm.contextualize (module), 18
PyOpenWorm.data (module), 19
PyOpenWorm.data_trans (module), 4
PyOpenWorm.data_trans.common_data (module), 4
PyOpenWorm.data_trans.connections (module), 4
PyOpenWorm.data_trans.csv_ds (module), 6
PyOpenWorm.data_trans.data_with_evidence_ds (module), 6
PyOpenWorm.data_trans.local_file_ds (module), 7
PyOpenWorm.data_trans.neuron_data (module), 7
PyOpenWorm.data_trans.wormatlas (module), 8
PyOpenWorm.data_trans.wormbase (module), 9
PyOpenWorm.dataObject (module), 21
PyOpenWorm.dataObjectUtils (module), 23
PyOpenWorm.datasource (module), 23
PyOpenWorm.document (module), 26
PyOpenWorm.documentContext (module), 27
PyOpenWorm.evidence (module), 27
PyOpenWorm.experiment (module), 28
PyOpenWorm.identifier_mixin (module), 29
PyOpenWorm.import_contextualizer (module), 29
PyOpenWorm.import_override (module), 30
PyOpenWorm.inverse_property (module), 30
PyOpenWorm.muscle (module), 30
PyOpenWorm.my_neuroml (module), 30
PyOpenWorm.network (module), 31
PyOpenWorm.neuron (module), 32
PyOpenWorm.package_utils (module), 35
PyOpenWorm.plot (module), 35
PyOpenWorm.pProperty (module), 34
PyOpenWorm.relationship (module), 35
PyOpenWorm.relationshipProxy (module), 36
PyOpenWorm.simpleProperty (module), 36
PyOpenWorm.statement (module), 36
PyOpenWorm.utils (module), 36
PyOpenWorm.website (module), 37

PyOpenWorm.worm (module), 37

Q

QueryContext (class in PyOpenWorm.context), 18

R

RDFSource (class in PyOpenWorm.data), 19
RealSimpleProperty (class in PyOpenWorm.simpleProperty), 36
reconstitute() (PyOpenWorm.command.POW method), 16
reference (PyOpenWorm.evidence.Evidence attribute), 28
refutes (PyOpenWorm.evidence.Evidence attribute), 28
Rel (class in PyOpenWorm.relationshipProxy), 36
Relationship (class in PyOpenWorm.relationship), 35
retract() (PyOpenWorm.dataObject.BaseDataObject method), 22
retract_statements() (PyOpenWorm.data.DataUser method), 19

S

save() (PyOpenWorm.dataObject.BaseDataObject method), 22
sensory() (PyOpenWorm.network.Network method), 32
SerializationSource (class in PyOpenWorm.data), 19
set() (PyOpenWorm.neuron.ConnectionProperty method), 32
set() (PyOpenWorm.neuron.Neighbor method), 33
set() (PyOpenWorm.pProperty.Property method), 34
set_data() (PyOpenWorm.plot.Plot method), 35
SleepyCatSource (class in PyOpenWorm.data), 20
SPARQLSource (class in PyOpenWorm.data), 20
Statement (class in PyOpenWorm.statement), 36
store_name (PyOpenWorm.command.POW attribute), 16
supports (PyOpenWorm.evidence.Evidence attribute), 28
Syn_degree() (PyOpenWorm.neuron.Neuron method), 34

T

tag() (PyOpenWorm.command.POW method), 16
title (PyOpenWorm.document.Document attribute), 26
translate() (PyOpenWorm.bibtex.BibTexDataTranslator method), 12
translate() (PyOpenWorm.command.POW method), 16
translate() (PyOpenWorm.data_trans.connections.NeuronConnectomeCSVTranslator method), 5
translate() (PyOpenWorm.data_trans.neuron_data.NeuronCSVDataTranslator method), 7
translate() (PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataTranslator method), 9
translate() (PyOpenWorm.data_trans.wormbase.MuscleWormBaseCSVTranslator method), 9
translate() (PyOpenWorm.data_trans.wormbase.NeuronWormBaseCSVTranslator method), 9
translate() (PyOpenWorm.data_trans.wormbase.WormbaseIonChannelCSV method), 10
translate() (PyOpenWorm.data_trans.wormbase.WormbaseTextMatchCSVT method), 11
translate() (PyOpenWorm.datasource.BaseDataTranslator method), 24
Translation (class in PyOpenWorm.datasource), 25
translation_type (PyOpenWorm.data_trans.connections.NeuronConnectomeCSVTranslator attribute), 5
translation_type (PyOpenWorm.data_trans.wormatlas.WormAtlasCellListDataTranslator attribute), 8
translation_type (PyOpenWorm.datasource.BaseDataTranslator attribute), 23
translation_type (PyOpenWorm.datasource.DataTranslator attribute), 25
TrixSource (class in PyOpenWorm.data), 20

U

UnionProperty (class in PyOpenWorm.simpleProperty), 36
UnionProperty() (PyOpenWorm.dataObject.BaseDataObject class method), 21
UnreadableGraphException, 15
update_from_wormbase() (PyOpenWorm.document.Document method), 26
uri (PyOpenWorm.document.Document attribute), 27

V

values (class in PyOpenWorm.dataObject), 23
variable() (PyOpenWorm.dataObject.BaseDataObject method), 22

W

wbid (PyOpenWorm.document.Document attribute), 27
Website (class in PyOpenWorm.website), 37
Worm (class in PyOpenWorm.worm), 37
WormAtlasCellListDataSource (class in PyOpenWorm.data_trans.wormatlas), 8
WormAtlasCellListDataTranslation (class in PyOpenWorm.data_trans.wormatlas), 8
WormAtlasCellListDataTranslator (class in PyOpenWorm.data_trans.wormatlas), 8
WormBaseCSVDataSource (class in PyOpenWorm.data_trans.wormbase), 9
wormbaseid (PyOpenWorm.document.Document attribute), 27
WormbaseIonChannelCSVDataSource (class in PyOpenWorm.data_trans.wormbase), 10

WormbaseIonChannelCSVTranslator (class in PyOpenWorm.data_trans.wormbase), 10
WormbaseRetrievalException, 26
WormbaseTextMatchCSVDataSource (class in PyOpenWorm.data_trans.wormbase), 10
WormbaseTextMatchCSVTranslator (class in PyOpenWorm.data_trans.wormbase), 11
write() (PyOpenWorm.my_neuroml.NeuroML class method), 30

Y

year (PyOpenWorm.document.Document attribute), 27

Z

ZODBSource (class in PyOpenWorm.data), 21