PyOpenWorm Documentation

Release alpha0.5

PyOpenWorm

Apr 25, 2019

Contents

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 Statement 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.

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

PyOpenWorm.**loadData**(*conf*, *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=None*, *do_logging=False*, *data=False*, *dataFormat='n3'*)

Load desired configuration and open the database

Parameters

- **configFile** – (Optional) The configuration file for PyOpenWorm
- **conf** – (Optional) a configuration object for the connection. Takes precedence over *configFile*
- **do_logging** – (Optional) If true, turn on debug level logging
- **data** – (Optional) specify the file to load into the library
- **dataFormat** – (Optional) file format of *data*. 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

Data translators

Some *DataSource* and *DataTranslator* types. Some deal with generic file types (e.g., comma-separated values) while others are specific to the format of a kind of file housed in PyOpenWorm.

Submodules

PyOpenWorm.data_trans.bibtex module

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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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(*self*)
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(*self*)
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**(***kwargs*)
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(*self*, *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(*self*, *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.context_datasource module

class PyOpenWorm.data_trans.context_datasource.**VariableIdentifierContext**(*maker=None*, ***kwargs*)

Bases: PyOpenWorm.data_trans.context_datasource.VariableIdentifierMixin, *PyOpenWorm.context.Context*

A Context that gets its identifier and its configuration from its 'maker' passed in at initialization

class PyOpenWorm.data_trans.context_datasource.**VariableIdentifierContextDataObject**(*maker=No*
***kwargs*)

Bases: PyOpenWorm.data_trans.context_datasource.VariableIdentifierMixin, *PyOpenWorm.contextDataObject.ContextDataObject*

A ContextDataObject that gets its identifier and its configuration from its 'maker' passed in at initialization

defined_augment(*self*)

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

PyOpenWorm.data_trans.context_merge module

class PyOpenWorm.data_trans.context_merge.**ContextMergeDataTranslator**(***kwargs*)

Bases: *PyOpenWorm.datasource.DataTranslator*

Input type(s): *PyOpenWorm.datasource.OneOrMore* (*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/ContextMergeDataTranslator

output_type

alias of *PyOpenWorm.data_trans.data_with_evidence_ds.DataWithEvidenceDataSource*

translate(*self*, **sources*)

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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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**(***kwargs*)

Bases: *PyOpenWorm.datasource.DataTranslator*

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

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

URI: None

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

Bases: *PyOpenWorm.data_trans.http_ds.HTTPFileDataSource*

CSV field delimiter [DatatypeProperty] Attribute: *csv_field_delimiter*

Header column names [DatatypeProperty] Attribute: *csv_header*

URL [DatatypeProperty] Attribute: *url*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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.data_with_evidence_ds module

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

Bases: *PyOpenWorm.datasource.DataSource*

Combined context [ObjectProperty] Attribute: *combined_context*

Context importing both the data and evidence contexts

Evidence context [ObjectProperty] Attribute: *evidence_context*

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

Data context [ObjectProperty] 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.excel_ds module

class PyOpenWorm.data_trans.excel_ds.**XLSXHTTPFileDataSource**(***kwargs*)

Bases: *PyOpenWorm.data_trans.http_ds.HTTPFileDataSource*

URL [DatatypeProperty] Attribute: *url*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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.file_ds module

class PyOpenWorm.data_trans.file_ds.**FileDataSource**(***kwargs*)

Bases: *PyOpenWorm.datasource.DataSource*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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.http_ds module

class PyOpenWorm.data_trans.http_ds.**HTTPFileDataSource**(***kwargs*)
Bases: *PyOpenWorm.data_trans.file_ds.FileDataSource*

URL [DatatypeProperty] Attribute: *url*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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.capability.Capable, *PyOpenWorm.data_trans.file_ds.FileDataSource*

File paths should be relative – in general, path names on a given machine are not portable

accept_capability_provider(*self*, *cap*, *provider*)
The Capable should replace any previously accepted provider with the one given.

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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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(*self*, *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.sci_rna_seq module

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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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(*self*)
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(*self*)
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**(***kwargs*)
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(*self*, *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(*self*, *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**(***kwargs*)
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(*self*, *data_source*)
Upload muscles and the neurons that connect to them

class PyOpenWorm.data_trans.wormbase.**NeuronWormBaseCSVTranslator**(***kwargs*)
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(*self*, *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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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**(***kwargs*)

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(*self*, *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**(***kwargs*)

Bases: *PyOpenWorm.data_trans.csv_ds.CSVDataSource*

initial_cell_column [DatatypeProperty] Attribute: *initial_cell_column*

The index of the first column with a cell name

cell_type [DatatypeProperty] Attribute: *cell_type*

The type of cell to be produced

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*

SHA-256 hash [DatatypeProperty] Attribute: *sha256*

SHA-512 hash [DatatypeProperty] Attribute: *sha512*

MD5 hash [DatatypeProperty] Attribute: *md5*

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**(***kwargs*)
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(*self*, *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

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

PyOpenWorm.bibtex_customizations module

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

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

Returns dict – the modified record.

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

Parameters **record** – a record

Returns – customized record

PyOpenWorm.bibtex_customizations.**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.capabilities module

class PyOpenWorm.capabilities.**FilePathCapability**
Bases: PyOpenWorm.capability.Capability

Provides a file path.

PyOpenWorm.capability module

Defines 'capabilities', pieces of functionality that an object needs which must be injected.

A given capability can be provided by more than one capability provider, but, for a given set of providers, only one will be bound at a time. Logically, each provider that provides the capability is asked, in a user-provided preference order, whether it can provide the capability for the *specific* object and the first one which can provide the capability is bound to the object.

The core idea is dependency injection: a capability does not modify the object: the object receives the provider and an identifier for the capability provided, but how the object uses the provider is up to the object. This is important because the user of the object should not condition its behavior on the particular capability provider used, although it may know about which capabilities the object has.

Note, that there may be some providers that lose their ability to provide a capability. This loss should be communicated with a 'CannotProvideCapability' exception when the relevant methods are called on the provider. This *may* allow certain operations to be retried with a provider lower on the capability order, *but* a provider that throws CannotProvide may validly be asked if it can provide the capability again – if it *still* cannot provide the capability, it should communicate that when asked.

Providers may keep state between calls to provide a capability, but their correctness must not depend on any ordering of method calls except that, of course, their __init__ is called first.

exception PyOpenWorm.capability.**CannotProvideCapability**(*cap*, *provider*)
Bases: exceptions.Exception

Thrown by a *provider* when it cannot provide the capability during the object's execution

exception PyOpenWorm.capability.**NoProviderAvailable**(*cap*, *receiver=None*)
Bases: exceptions.Exception

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(*self*)
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(*self*)
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(*self*, **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.cell_common module

PyOpenWorm.channel module

class PyOpenWorm.channel.**Channel**(*name=None*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

A biological ion channel.

Attributes

Models [Property]

defined_augment(*self*)
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(*self*)
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

appearsIn
Cell types in which the ion channel has been expressed

description
A description of the ion channel

expression_pattern
A pattern of expression of this cell within an organism

gene_WB_ID
Wormbase ID of the encoding gene

gene_class
Classification of the encoding gene

gene_name
Name of the gene that codes for this ion channel

model
Get experimental models of this ion channel

name
Ion channel's name

neuroml_file
A NeuroML describing a model of this ion channel

proteins
Proteins associated with this channel

subfamily
Ion channel's subfamily

class PyOpenWorm.channel.**ExpressionPattern**(*wormbaseid=None*, ***kwargs*)
Bases: *PyOpenWorm.biology.BiologyType*

defined_augment(*self*)
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(*self*)
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
Natural language description of the expression pattern

wormbaseID
Alias to *wormbaseid*

wormbaseURL
The URL for the expression pattern in Wormbase

wormbaseid
The ID for the expression pattern in Wormbase

PyOpenWorm.channel_common module

PyOpenWorm.channel_common.**CHANNEL_RDF_TYPE = rdflib.term.URIRef(u'http://openworm.org/entit**
Shared RDF type for channels

PyOpenWorm.channelworm module

class PyOpenWorm.channelworm.**ChannelModel**(*modelType=None*, **args*, ***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

A model for an ion channel.

There may be multiple models for a single channel.

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()
```

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

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

ion
The type of ion this channel selects for

modelType
The type of model employed to describe a channel

class PyOpenWorm.channelworm.**HomologyChannelModel**(***kwargs*)
Bases: *PyOpenWorm.channelworm.ChannelModel*

class PyOpenWorm.channelworm.**PatchClampChannelModel**(***kwargs*)
Bases: *PyOpenWorm.channelworm.ChannelModel*

class PyOpenWorm.channelworm.**PatchClampExperiment**(***kwargs*)
Bases: *PyOpenWorm.experiment.Experiment*

Store experimental conditions for a patch clamp experiment.

Ca_concentration
Calcium concentration

Cl_concentration
Chlorine concentration

blockers
Channel blockers used for this experiment

cell
The cell this experiment was performed on

cell_age
Age of the cell

initial_voltage
Starting voltage of the patch clamp

ion_channel
The ion channel being clamped

membrane_capacitance
Initial membrane capacitance

mutants
Type(s) of mutants being used in this experiment

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

pipette_solution
Type of solution in the pipette

PyOpenWorm.cli module

PyOpenWorm.cli.**additional_args**(*parser*)
Add some additional options specific to CLI

PyOpenWorm.cli_command_wrapper module

exception PyOpenWorm.cli_command_wrapper.**CLIUserError**
Bases: exceptions.Exception

class PyOpenWorm.cli_command_wrapper.**CLIAppendAction**(*mapper*, *key*, *index=-1*, **args*, ***kwargs*)
Bases: *PyOpenWorm.cli_command_wrapper.CLIStoreAction*

class PyOpenWorm.cli_command_wrapper.**CLIArgMapper**
Bases: object

Stores mappings for arguments and maps them back to the part of the object they come from

runners = None
Mapping from subcommand names to functions which run for them

class PyOpenWorm.cli_command_wrapper.**CLIStoreAction**(*mapper*, *key*, *index=-1*, **args*, ***kwargs*)
Bases: argparse.Action

Interacts with the CLIArgMapper

class PyOpenWorm.cli_command_wrapper.**CLIStoreTrueAction**(**args*, ***kwargs*)
Bases: *PyOpenWorm.cli_command_wrapper.CLIStoreAction*

class PyOpenWorm.cli_command_wrapper.**CLISubCommandAction**(*mapper*, **args*, ***kwargs*)

Bases: argparse._SubParsersAction

PyOpenWorm.cli_common module

PyOpenWorm.cli_hints module

PyOpenWorm.collections module

class PyOpenWorm.collections.**Bag**(***kwargs*)

Bases: *PyOpenWorm.dataObject.DataObject*

A convenience class for working with a collection of objects

Example:

```
v = Bag('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 = Bag('unc-13 neurons and muscles')
nm = list(u.value())
```

defined_augment(*self*)

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(*self*)

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

add

An alias for value

group_name

Alias for name

name

The name of the group of objects

value

An object in the group

PyOpenWorm.command module

exception PyOpenWorm.command.**ConfigMissingException**(*key*)
Bases: *PyOpenWorm.command.GenericUserError*

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

An error which should be reported to the user. Not necessarily an error that is the user's fault

exception PyOpenWorm.command.**InvalidGraphException**
Bases: *PyOpenWorm.command.GenericUserError*

Thrown when a graph cannot be translated due to formatting errors

exception PyOpenWorm.command.**NoConfigFileError**
Bases: *PyOpenWorm.command.GenericUserError*

exception PyOpenWorm.command.**POWDirMissingException**
Bases: *PyOpenWorm.command.GenericUserError*

exception PyOpenWorm.command.**StatementValidationError**(*statements*)
Bases: *PyOpenWorm.command.GenericUserError*

exception PyOpenWorm.command.**UnreadableGraphException**
Bases: *PyOpenWorm.command.GenericUserError*

Thrown when a graph cannot be read due to it being missing, the active user lacking permissions, etc.

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

High-level commands for working with PyOpenWorm data

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

Parameters

url [str] URL of the data store to clone

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

commit(*self*, *message*)
Write the graph to the local repository

Parameters

message [str] commit message

context(*self*, *context=None*, *user=False*)
Read or set current target context for the repository

Parameters

context [str] The context to set

user [bool] If set, set the context only for the current user. Has no effect for retrieving the context

diff(*self*)
Show differences between what's in the working context set and what's in the serializations

fetch_graph(*self*, *url*)
Fetch a graph

Parameters

url [str] URL for the graph

git(*self*, **args*)
Runs git commmands in the .pow directory

Parameters

***args** [*str] arguments to git

imports_context(*self*, *context=None*, *user=False*)
Read or set current target imports context for the repository

Parameters

context [str] The context to set

user [bool] If set, set the context only for the current user. Has no effect for retrieving the context

init(*self*, *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

list_contexts(*self*)
List contexts

merge(*self*)

push(*self*)

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

Parameters

data_source [str] Identifier for the data source to reconstitute

save(*self*, *module*, *provider=None*, *context=None*)
Save the data in the given context

Parameters

module [str] Name of the module housing the provider

provider [str] Name of the provider, a callble that accepts a context object and adds statements to it. Can be a "dotted" name indicating attribute accesses

context [str] The target context

serialize(*self*, *context=None*, *destination=None*, *format='nquads'*, *whole_graph=False*)
Serialize the current data context or the one provided

Parameters

context [str] The context to save

destination [file or str] A file-like object to write the file to or a file name. If not provided, messages the result.

format [str] Serialization format (ex, 'n3', 'nquads')

whole_graph: bool Serialize all contexts from all graphs (this probably isn't what you want)

tag(*self*)

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

Parameters

translator [str] Translator identifier

imports_context_ident [str] Identifier for the imports context. All imports go in here

output_key [str] Output key. Used for generating the output's identifier. Exclusive with output_identifier

output_identifier [str] Output identifier. Exclusive with output_key

data_sources [list of str] Input data sources

named_data_sources [dict] Named input data sources

config_file
The config file name

log_level
Log level

powdir
The base directory for PyOpenWorm files. The repository provider's files also go under here

store_name
The file name of the database store

class PyOpenWorm.command.**POWSource**(*parent*)
Bases: object

Commands for working with DataSource objects

create(*self*, *kind*, *key*, **args*, ***kwargs*)
Create the source and add it to the graph.

Arguments are determined by the type of the data source

Parameters

kind [rdflib.term.URIRef] The kind of source to create

key [str] The key, a unique name for the source

list(*self*, *context=None*, *kind=None*, *full=False*)
List known sources

Parameters

kind [str] Only list sources of this kind

context [str] The context to query for sources

full [bool] Whether to (attempt to) shorten the source URIs by using the namespace manager

list_kinds(*self*, *full=False*)
List kinds of sources

Parameters

full [bool] Whether to (attempt to) shorten the source URIs by using the namespace manager

show(*self*, **data_source*)

Parameters

***data_source** [str] The ID of the data source to show

data
Commands for saving and loading data for DataSources

class PyOpenWorm.command.**POWSourceData**(*parent*)
Bases: object

Commands for saving and loading data for DataSources

retrieve(*self*, *source*, *archive='data.tar'*, *archive_type=None*)
Retrieves the data for the source

Parameters

source [str] The source for data

archive [str] The file name of the archive. If this ends with an extension like '.zip', and no *archive_type* argument is given, then an archive will be created of that type. The archive name will *not* have any extension appended in any case.

archive_type [str] The type of the archive to create.

class PyOpenWorm.command.**SaveValidationFailureRecord**
Bases: *PyOpenWorm.command.SaveValidationFailureRecord*

class PyOpenWorm.command.**UnimportedContextRecord**
Bases: *PyOpenWorm.command.UnimportedContextRecord*

PyOpenWorm.command_util module

class PyOpenWorm.command_util.**IVar**(*default_value=None*, *doc=None*, *value_type=<type 'str'>*, *name=None*)
Bases: object

A descriptor for instance variables amended to provide some attributes like default values, value types, etc.

class PyOpenWorm.command_util.**PropertyIVar**(**args*, ***kwargs*)
Bases: *PyOpenWorm.command_util.IVar*

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

Unlike a *dict*, Configure objects will execute a function when retrieving values to enable deferred computation of seldom-used configuration values. In addition, entries in a *Configure* can be aliased to one another.

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

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

Returns

get(*self*, *pname*, *default=NO_DEFAULT*)
Get some parameter value out by asking for a key. Note that unlike dict, if you don't specify a default, then a KeyError is raised

Parameters

pname [str] they key of the value you want to return.

default [object] The default value to return if there's no entry for *pname*

Returns

The value corresponding to the key

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

classmethod open(*cls*, *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(*self*, *pname*, *default=None*)
Gets a config value from this *Configureable*'s *conf*

See also:

Configure.get

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*

defined_augment(*self*)

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(*self*)

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

number

The weight of the connection

post_cell

The post-synaptic cell

pre_cell

The pre-synaptic cell

synclass

The kind of Neurotransmitter (if any) sent between *pre_cell* and *post_cell*

syntype

The kind of synaptic connection. 'gapJunction' indicates a gap junction and 'send' a chemical synapse

termination

Where the connection terminates. Inferred from type of post_cell at initialization

PyOpenWorm.context module

class PyOpenWorm.context.**ClassContext**(*key=None*, *imported=()*, *ident=None*, *mapper=None*, *base_namespace=None*, ***kwargs*)

Bases: *PyOpenWorm.context.Context*

class PyOpenWorm.context.**ClassContextMeta**

Bases: *PyOpenWorm.context.ContextMeta*

class PyOpenWorm.context.**Context**(*key=None*, *imported=()*, *ident=None*, *mapper=None*, *base_namespace=None*, ***kwargs*)

Bases: *PyOpenWorm.import_contextualizer.ImportContextualizer*, *PyOpenWorm.context.ContextualizableDataUserMixin*

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

save(*self*, *graph=None*, *inline_imports=False*, *autocommit=True*, *saved_contexts=None*)

Alias to save_context

save_context(*self*, *graph=None*, *inline_imports=False*, *autocommit=True*, *saved_contexts=None*)

Save the context to a graph

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.**ContextualizableDataUserMixin**(**args*, ***kwargs*)
Bases: *PyOpenWorm.contextualize.Contextualizable*, *PyOpenWorm.data.DataUser*

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(*self*)
Close a the configured database

init_database(*self*)
Open the configured database

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

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

classmethod process_config(*cls*, *config_dict*)
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(*self*, *g*, *reference_iri*)
Add a citation to a set of statements in the database

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

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

Parameters **graph** – An iterable of triples

infer(*self*)
Fire FuXi rule engine to infer triples

retract_statements(*self*, *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(*self*)
Called on `PyOpenWorm.connect()` to set up and return the rdflib graph. Must be overridden by sub-classes.

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

Reads from and queries against a remote data store

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

open(*self*)
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(*self*)
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(*self*)
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(*self*)
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.context.ContextualizableDataUserMixin*

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(*cls*, **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(*cls*, **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(*cls*, **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(*self*)
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(*self*)
Return the object with all contexts removed

get_owners(*self*, *property_class_name*)
Return a generator of owners along a property pointing to this object

graph_pattern(*self*, *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(*self*)
Is the identifier a variable?

retract(*self*)
Remove this object from the data store.

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

variable(*self*)
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__ method definition, then no special processing is done.

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

after_mapper_module_load(*self*, *mapper*)
Called after the module has been loaded. See *PyOpenWorm.mapper.Mapper*

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

query
Stub. Eventually, creates a proxy that changes how some things behave for purposes of querying

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

PyOpenWorm.datasource module

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

class PyOpenWorm.datasource.**BaseDataTranslator**(***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

Translates from a data source to PyOpenWorm objects

input_type
alias of *DataSource*

output_type
alias of *DataSource*

translation_type
alias of *Translation*

make_translation(*self*, *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(*self*, **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.DataObject*

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__.

commit(*self*)
Commit the data source *locally*

This includes staging files such as they would be available for a translation. In general, a sub-class should implement `commit_augment()` rather than this method, or at least call this method via super

For example, if the data source produces a file, that file should be in

defined_augment(*self*)
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(*self*)
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**(***kwargs*)
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(*self*, *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**(***kwargs*)
Bases: *PyOpenWorm.datasource.Translation*

A generic translation that just has sources in order

defined_augment(*self*)
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(*self*)
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.**OneOrMore**(*source_type*)
Bases: object

Wrapper for *DataTranslator* input *DataSource* types indicating that one or more of the wrapped type must be provided to the translator

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

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

person
A person responsible for carrying out the translation.

class PyOpenWorm.datasource.**Translation**(***kwargs*)
Bases: *PyOpenWorm.dataObject.DataObject*

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(*self*)
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(*self*)
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.datasource_loader module

DataSourceLoaders take a data source identifier and retrieve the primary data (e.g., CSV files, electrode recordings) from some location (e.g., a file store, via a bittorrent tracker).

Each loader can treat the base_directory given as its own namespace and place directories in there however it wants.

exception PyOpenWorm.datasource_loader.**LoadFailed**(*ident*, *loader*, **args*)
Bases: exceptions.Exception

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(*self*)
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(*self*)
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(*self*, *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(*self*)
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(*self*)
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.evidence.**evidence_for**(*qctx*, *ctx*, *evctx=None*)

Returns an iterable of Evidence

Parameters

qctx [object] an object supported by evidence. If the object is a *Context* with no identifier, then the query considers statements 'staged' (rather than stored) in the context

ctx [Context] Context that bounds where we look for statements about *qctx*. The contexts for statements found in this context are the actual targets of Evidence.supports statements.

evctx [Context] if the Evidence.supports statements should be looked for somewhere other than *ctx*, that can be specified in evctx. optional

PyOpenWorm.evidence.**query_context**(*graph*, *qctx*)

graph [rdflib.graph.Graph] Graph where we can find the contexts for statements in *qctx*

qctx [PyOpenWorm.context.Context] Container for statements

PyOpenWorm.experiment module

class PyOpenWorm.experiment.**Experiment**(***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.

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

reference
Supporting article for this experiment.

PyOpenWorm.git_repo module

PyOpenWorm.identifier_mixin module

PyOpenWorm.identifier_mixin.**IdMixin**(*typ=<type 'object'>*, *hashfunc=None*)
Mixin that provides common identifier logic

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.mapper module

exception PyOpenWorm.mapper.**UnmappedClassException**
Bases: exceptions.Exception

class PyOpenWorm.mapper.**Mapper**(*base_class_names*, *base_namespace=None*, *imported=()*, *name=None*, ***kwargs*)
Bases: PyOpenWorm.module_recorder.ModuleRecordListener, *PyOpenWorm.configure.Configureable*

Keeps track of relationships between classes, between modules, and between classes and modules

decorate_class(*self*, *cls*)
Extension point for subclasses of Mapper to apply an operation to all mapped classes

load_module(*self*, *module_name*)
Loads the module.

lookup_class(*self*, *cname*)
Gets the class corresponding to a fully-qualified class name

DecoratedMappedClasses = None
Maps RDF types to properties of the related class

MappedClasses = None
Maps classes to decorated versions of the class

base_modules = None
The base class for objects that will be mapped.

Defined once the module containing the class is loaded

base_namespace = None
Modules that have already been loaded

PyOpenWorm.module_recorder module

PyOpenWorm.muscle module

class PyOpenWorm.muscle.**Muscle**(*name=None*, *lineageName=None*, ***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
```

innervatedBy
Neurons synapsing with this muscle

neurons
Alias to *innervatedBy*

receptor
Alias to *receptors*

receptors
Receptor types expressed by this type of muscle

PyOpenWorm.my_neuroml module

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

classmethod generate(*cls*, *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(*cls*, *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

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

Example:

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

# Grab a specific neuron
```

(continues on next page)

(continued from previous page)

```
>>> 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(*self*)
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(*self*)
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(*self*)
Get all interneurons

Returns A iterable of all interneurons

Return type iter(*Neuron*)

motor(*self*)
Get all motor

Returns A iterable of all motor neurons

Return type iter(*Neuron*)

neuron_names(*self*)
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(*self*)
Get all sensory neurons

Returns A iterable of all sensory neurons

Return type iter(*Neuron*)

neuron
Returns a set of all Neuron objects in the network

neurons
Alias to *neuron*

synapse
Returns a set of all synapses in the network

synapses
Alias to *synapse*

worm
The worm connected to the network

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(*self*, *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(*self*, *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(*self*, ***kwargs*)
Get a list of neighboring neurons.

Parameters

See parameters for PyOpenWorm.connection.Connection

Returns

list of Neuron

set(*self*, *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

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(*self*)

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(*self*)

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(*self*, *type=0*)

Get neurons which synapse at this neuron

innexin
Innexin types associated with this neuron

neuropeptide
Name of the gene corresponding to the neuropeptide produced by this neuron

neurotransmitter
Neurotransmitters associated with this neuron

receptor
The receptor types associated with this neuron

receptors
Alias to receptor

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

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(*self*, **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(*self*)
Returns true if the Property has any values set on it.

This may be defined differently for each property

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

set(*self*, **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=None*, **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(*self*)
Get the data stored for this plot.

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

PyOpenWorm.rdf_go_modifiers module

PyOpenWorm.rdf_query_util module

PyOpenWorm.rdf_query_util.**get_most_specific_rdf_type**(*types*, *context=None*, *bases=()*)
Gets the most specific rdf_type.

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

PyOpenWorm.rdf_query_util.**oid**(*identifier_or_rdf_type=None*, *rdf_type=None*, *context=None*, *base_type=None*)
Create an object from its rdf type

Parameters

identifier_or_rdf_type [str or rdflib.term.URIRef] If *rdf_type* is provided, then this value is used as the identifier for the newly created object. Otherwise, this value will be the rdf_type of the object used to determine the Python type and the object's identifier will be randomly generated.

rdf_type [str, rdflib.term.URIRef, False] If provided, this will be the rdf_type of the newly created object.

Returns

The newly created object

PyOpenWorm.simpleProperty module

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

class PyOpenWorm.simpleProperty.**ContextualizedPropertyValue**(*value*)
Bases: yarom.propertyValue.PropertyValue

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

class PyOpenWorm.simpleProperty.**ObjectProperty**(*resolver=None*, **args*, ***kwargs*)
Bases: *PyOpenWorm.inverse_property.InversePropertyMixin*, PyOpenWorm.simpleProperty._ContextualizingPropertySetMixin, 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(*self*)
Clears values set *in all contexts*

decontextualize(*self*)
Return the object with all contexts removed

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**(*title=None*, ***kwargs*)
Bases: *PyOpenWorm.document.BaseDocument*

A representation of a website

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

identifier_augment(*self*)
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

title
The official name for the website

url
A URL for the website

PyOpenWorm.worm module

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

A representation of the whole worm

defined_augment(*self*)
True if the name is defined

get_neuron_network(*self*)
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'])

#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(*self*)

Get the underlying semantic network as an RDFLib Graph

Returns A semantic network containing information about the worm

Return type rdflib.ConjunctiveGraph

identifier_augment(*self*, **args*, ***kwargs*)
Result is derived from the name property

muscles(*self*)
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`

cell
A cell in the worm

muscle
A type of muscle which is in the worm

name
Alias to *scientific_name*

neuron_network
The neuron network of the worm

scientific_name
Scientific name for the organism

PyOpenWorm.worm_common module

CHAPTER 2

For Users

2.1 PyOpenWorm Data Sources

The sources of data for PyOpenWorm are stored in the **'OpenWorm-Data repository<https://github.com/openworm/PyOpenWorm>'_**. A few **:py:class:'DataTranslators<~PyOpenWorm.datasource.DataTranslator>'_** translate these data into common PyOpenWorm data sources. You can list these by running:

```
pow source list
```

and you can show some of the properties of a data source by running:

```
pow source show $SOURCE_IDENTIFIER
```

For instance, you can run the following to see the top-level data source, try:

```
pow source show http://openworm.org/data
```

This will print out summary descriptions of the sources that contribute to the main data source.

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.
- 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]
- 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

[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

2.1.4 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.

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.

[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

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!

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:

```
>>> from PyOpenWorm.neuron import Neuron
>>> n = Neuron.query()
>>> n.receptor('UNC-13')
PyOpenWorm.statement.Statement(...obj=yarom.propertyValue.PropertyValue(rdflib.term.
↪Literal(u'UNC-13')), context=None)
>>> for x in n.load():
...     do_something_with_unc13_neuron(n)  # doctest.SKIP
```

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:

```
>>> from PyOpenWorm.context import Context
>>> from PyOpenWorm.worm import Worm
>>> from PyOpenWorm.network import Network
>>> 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
PyOpenWorm.statement.Statement(...)
>>> n = ctx(Neuron)('NeuronX')   # make a neuron
>>> n.receptor('UNC-13')         # state that the neuron has a UNC-13 type receptor
PyOpenWorm.statement.Statement(...)
>>> nn.neuron(n)                 # attach to the neuron network
```

(continues on next page)

(continued from previous page)

```
PyOpenWorm.statement.Statement(...)
>>> 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 made all of the statements, you must save the context in which the statements are made.

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)
    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 used contexts without explaining them. In natural languages, our statements are made in a context that influences how they should be interpreted. In PyOpenWorm, that kind of context-sensitivity is modeled by using *PyOpenWorm.context.Context* objects. To see what this looks like, let's start with an example.

Basics

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 own POW widget model
>>> # 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,
...             hardiness=record.hrds)
Widget(ident=rdflib.term.URIRef(...))


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

>>> # Save the data
>>> ctx.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').decode('UTF-8'))
<http://openworm.org/entities/Widget/12> <http...> <http://example.org/data/imports/
↪BDW_Widgets_2017-2018> .
<http://openworm.org/entities/Widget/12> <...
```

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`). These additional contexts could then have separate metadata attached to them or they could be compared:

```
>>> len(list(ctx(Widget)().load()))
1
>>> len(list(ctx18(Widget)().load()))  # 2018-2019 context
3
```

Context Metadata

Contexts, because they have identifiers just like any other objects, so we can make statements about them as well. An essential statement is imports: Contexts import other contexts, which means, if you follow PyOpenWorm semantics, that when you query objects from the importing context, that the imported contexts will also be available to query.

2.4 Making data objects

To make a new object type like *Neuron* or *PyOpenWorm.worm.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.dataObject import (DataObject,
...                                    DatatypeProperty,
...                                    ObjectProperty,
...                                    Alias)
>>> from PyOpenWorm.worm import Worm
>>> from PyOpenWorm.evidence import Evidence
>>> from PyOpenWorm.document import Document
>>> from PyOpenWorm.context import Context
>>> from PyOpenWorm.mapper import Mapper
>>> from PyOpenWorm import connect, ModuleRecorder

>>> class Drug(DataObject):
...     name = DatatypeProperty()
...     drug_name = Alias(name)
...     def identifier_augment(self):
...         return self.make_identifier_direct(self.name.onedef())
...
...     def defined_augment(self):
...         return self.name.has_defined_value()

>>> class Experiment(DataObject):
...     drug = ObjectProperty(value_type=Drug)
...     subject = ObjectProperty(value_type=Worm)
...     route_of_entry = DatatypeProperty()
...     reaction = DatatypeProperty()

# Do some accounting stuff to register the classes. Usually happens behind
# the scenes.
>>> m = Mapper(('PyOpenWorm.dataObject.DataObject',))
>>> ModuleRecorder.add_listener(m)
>>> m.process_classes(Drug, Experiment)
```

So, we have created I can then make a Drug object for moon rocks and describe an experiment by Aperture Labs:

```
>>> ctx = Context(ident='http://example.org/experiments', mapper=m)
>>> d = ctx(Drug)(name='moon rocks')
>>> e = ctx(Experiment)(key='experiment001')
>>> w = ctx(Worm)('C. elegans')
>>> e.subject(w)
PyOpenWorm.statement.Statement(...Context(.../experiments"))

>>> e.drug(d)
```

(continues on next page)

(continued from previous page)

```
PyOpenWorm.statement.Statement(...)

>>> e.route_of_entry('ingestion')
PyOpenWorm.statement.Statement(...)

>>> e.reaction('no reaction')
PyOpenWorm.statement.Statement(...)

>>> ev = Evidence(key='labresults', reference=Document(author="Aperture Labs"))
>>> ev.supports(ctx)
PyOpenWorm.statement.Statement(...)
```

and save those statements:

```
>>> ctx.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 class:

```
>>> d = ctx(Drug)(name='moon rocks')
>>> Drug.DatatypeProperty('granularity', owner=d)
__main__.Drug_granularity(owner=Drug(ident=rdflib.term.URIRef(u'http://openworm.org/
↪entities/Drug/moon%20rocks')))

>>> d.granularity('ground up')
PyOpenWorm.statement.Statement(...Context(.../experiments"))

>>> do = Drug()
```

Properties added in this fashion will not propagate to any other objects:

```
>>> do.granularity
Traceback (most recent call last):
    ...
AttributeError: 'Drug' object has no attribute 'granularity'
```

They will, however, 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.

2.6 Working with contexts

Contexts are introduced in *Adding Data to YOUR OpenWorm Database*. Here we provide a little more. . . context.

2.6.1 Background

Contexts were introduced to PyOpenWorm (POW) as a generic tool for grouping statements. We need to group statements to make statements about statements like "Who made these statements?" or "When were these statements made?". That's the main usage. Beyond that, we need a way to share statements. Contexts have identifiers by which we can naturally refer to contexts from other contexts.

POW need a way to represent contexts with the existing statement form. Other alternatives were considered, such as using Python's context managers, but I (Mark) also wanted a way to put statements in a context that could also be carried with the subject of the statement. Using the wrapt package's proxies allows to achieve this while keeping the interface of the wrapped object the same, which is useful since it doesn't require a user of the object to know anything about contexts unless they need to change the context of a statement.

The remainder of this page will go into doing some useful things with contexts.

2.6.2 Classes and contexts

POW can load classes as well as instances from an RDF graph. The packages which define the classes must already be installed in the Python library path, and a few statements need to be in the graph you are loading from or in a graph imported (transitively) by that graph. The statements you need are these

```
:a_class_desc <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://openworm.org/
↪entities/PythonClassDescription> .
:a_class_desc <http://openworm.org/entities/ClassDescription/module> :a_module .
:a_class_desc <http://openworm.org/entities/PythonClassDescription/name> "AClassName"␣
↪.

:a_module <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://openworm.org/
↪entities/PythonModule> .
:a_module <http://openworm.org/entities/PythonModule/name> "APackage.and.module.name"␣
↪.
```

(continues on next page)

(continued from previous page)

where `:a_class_desc` and `:a_module` are placeholders for objects which will typically be created by POW on the user's behalf, and `AClassName` is the name of the class available at the top-level of the module `APackage.and.module.name`. These statements will be created in memory by POW when a module defining a *DataObject*-derived class is first processed by a *Mapper* which will happen after the module is imported.

CHAPTER 3

For Developers

3.1 Testing in PyOpenWorm

3.1.1 Preparing for tests

PyOpenWorm should be installed like:

```
python setup.py develop
```

The default database should be populated like:

```
pow clone https://github.com/openworm/OpenWormData.git
```

3.1.2 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'
```

3.1.3 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 PEP-8 Guidelines for contributions of Python code to PyOpenWorm, with some exceptions noted below. 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,E128
```

Refer to the pep8 documentation for the meanings of these error codes.

Lines of code should only be wrapped before 120 chars for readability. Comments and string literals, including docstrings, can be wrapped to a shorter length.

Some violations can be corrected with `autopep8`.

CHAPTER 4

Issues

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

L

M

N

O

Q

R

S

T

U

V

W

X

Y

Z