owmeta Documentation

Release 0.11.3.dev0

owmeta

Sep 15, 2020

Contents

1 owmeta **3**

1.1 owmeta package 3

2 For Users **13**

2.1 owmeta Data Sources 13

2.2 Requirements for data storage in OpenWorm 15

2.3 Adding Data to *YOUR* OpenWorm Database 17

2.4 Software Versioning 20

2.5 Python Release Compatibility 20

3 For Developers **23**

3.1 Testing in owmeta 23

3.2 Adding documentation 24

3.3 owmeta coding standards 25

4 Issues **27**

5 Indices and tables **29**

Python Module Index **31**

Index **33**

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/owmeta

CHAPTER 1

owmeta

1.1 owmeta package

1.1.1 owmeta

OpenWorm Unified Data Abstract Layer.

An introduction to owmeta 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.

class owmeta.**Configurable**(*conf=None*, ***kwargs*)

Bases: `object`

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

get(*pname*, *default=None*)

Gets a config value from this *Configurable*'s `conf`

See also:

Configuration.get

1.1.2 Subpackages

owmeta.commands package

Various commands of the same kind as `OWM`, mostly intended as sub-commands of `OWM`.

Submodules

owmeta.commands.biology module

owmeta.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 owmeta.

Submodules

owmeta.data_trans.bibtex module

class `owmeta.data_trans.bibtex.`**`BibTexDataSource`**(*bibtex_file_name*, ***kwargs*)

Bases: `owmeta.data_trans.common_data.DSMixin`, `owmeta_core.data_trans.local_file_ds.LocalFileDataSource`

File name [`DatatypeProperty`] Attribute: `file_name`

Torrent file name [`DatatypeProperty`] Attribute: `torrent_file_name`

MD5 hash [`DatatypeProperty`] Attribute: `md5`

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

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

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

Description [`DatatypeProperty`] Attribute: `description`

Free-text describing the data source

class `owmeta.data_trans.bibtex.`**`BibTexDataTranslator`**(***kwargs*)

Bases: `owmeta.data_trans.common_data.DTMixin`, `owmeta_core.datasource.DataTranslator`

Input type(s): *`BibTexDataSource`*

Output type(s): *`EvidenceDataSource`*

`input_type`

alias of *`BibTexDataSource`*

`output_type`

alias of *`EvidenceDataSource`*

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

class owmeta.data_trans.bibtex.**EvidenceDataSource**(**args*, ***kwargs*)
Bases: owmeta.data_trans.common_data.DSMixin, owmeta_core.datasource.DataSource

Context [ObjectProperty] Attribute: evidence_context

The context

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

Description [DatatypeProperty] Attribute: description

Free-text describing the data source

owmeta.data_trans.common_data module

owmeta.data_trans.connections module

owmeta.data_trans.context_merge module

class owmeta.data_trans.context_merge.**ContextMergeDataTranslator**(***kwargs*)
Bases: owmeta.data_trans.common_data.DTMixin, owmeta_core.datasource.DataTranslator

Input type(s): owmeta_core.datasource.OneOrMore (*DataWithEvidenceDataSource*)

Output type(s): *DataWithEvidenceDataSource*

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

translate(**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.

owmeta.data_trans.data_with_evidence_ds module

class owmeta.data_trans.data_with_evidence_ds.**DataWithEvidenceDataSource**(**args*, ***kwargs*)
Bases: owmeta.data_trans.common_data.DSMixin, owmeta_core.datasource.DataSource

A data source that has an "evidence context" containing statements which support those in its "data context". The data source also has a combined context which imports both the data and evidence contexts.

owmeta.data_trans.neuron_data module

owmeta.data_trans.wormatlas module

owmeta.data_trans.wormbase module

1.1.3 Submodules

owmeta.bibtex module

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

owmeta.bibtex_customizations module

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

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

Returns dict – the modified record.

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

Parameters **`record`** – a record

Returns – customized record

`owmeta.bibtex_customizations.`**`doi`**`(`*record*`)`

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

Returns dict – the modified record.

owmeta.biology module

`class` `owmeta.biology.`**`BiologyType`**`(`***kwargs*`)`
Bases: `owmeta_core.dataobject.DataObject`

owmeta.cell module

owmeta.cell_common module

owmeta.channel module

owmeta.channel_common module

`owmeta.channel_common.`**`CHANNEL_RDF_TYPE = rdflib.term.URIRef('http://schema.openworm.org/202`**
Shared RDF type for channels

owmeta.channelworm module

owmeta.cli_hints module

owmeta.command module

owmeta.connection module

owmeta.document module

exception owmeta.document.**PubmedRetrievalException**

Bases: Exception

exception owmeta.document.**WormbaseRetrievalException**

Bases: Exception

class owmeta.document.**BaseDocument**(***kwargs*)

Bases: owmeta_core.dataobject.DataObject

class owmeta.document.**Document**(*bibtex=None*, *doi=None*, *pubmed=None*, *wormbase=None*, ***kwargs*)

Bases: *owmeta.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
```

Parameters

- **bibtex** [str] A string containing a single BibTeX entry. Parsed during initialization, but not saved thereafter. optional
- **doi** [str] A Digital Object Identifier (DOI). optional
- **pubmed** [str] A PubMed ID (PMID) or URL that points to a paper. Ignored if ‘pmid’ is provided. optional
- **wormbase** [str] An ID or URL from WormBase that points to a record. Ignored if *wbid* or *wormbaseid* are provided. optional

defined_augment()

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

identifier_augment()

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

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

Raises

IdentifierMissingException

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

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

author
An author of the document

date
Alias to year

doi
A Digital Object Identifier (DOI), optional

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

title
The title of the document

uri
A non-standard URI for the document

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

wormbaseid
An alias to *`wbid`*

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

owmeta.documentContext module

class `owmeta.documentContext.`**DocumentContext**(*document*)
Bases: `owmeta_core.context.Context`

A Context that corresponds to a document.

class `owmeta.documentContext.`**DocumentContextMeta**
Bases: `owmeta_core.context.ContextMeta`

owmeta.evidence module

exception `owmeta.evidence.`**EvidenceError**
Bases: `Exception`

class `owmeta.evidence.`**Evidence**(***kwargs*)
Bases: `owmeta_core.dataobject.DataObject`

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

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

```
>>> from owmeta.connection import Connection
>>> from owmeta.evidence import Evidence
>>> from owmeta_core.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 the statement is sufficient.

`defined_augment``()`

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

`identifier_augment``()`

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

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

Raises

`IdentifierMissingException`

`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

`owmeta.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

`owmeta.evidence.`**`query_context`**`(`*graph*, *qctx*`)`

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

qctx [owmeta.context.Context] Container for statements

owmeta.experiment module

`class` `owmeta.experiment.`**`Experiment`**`(`***kwargs*`)`

Bases: `owmeta_core.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``()`

Return conditions and their associated values in a dict.

`reference`

Supporting article for this experiment.

owmeta.muscle module

owmeta.my_neuroml module

owmeta.network module

owmeta.neuroml module

`class` `owmeta.neuroml.`**`NeuroMLDocument`**`(`***kwargs*`)`

Bases: `owmeta_core.dataobject.DataObject`

Represents a NeuroML document

The document may be represented literally in the RDF graph using `xml_content` or stored elsewhere and included by reference with *`document_url`*.

Example:

```
>>> embedded_nml = NeuroMLDocument(key='embedded_ex', content="""\
... <?xml version="1.0" encoding="UTF-8"?>
... <neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
...     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
...     xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2
...     https://raw.github.com/NeuroML/NeuroML2/master/Schemas/NeuroML2/NeuroML_
↪v2beta.xsd"
...     id="k_slow">
...     <ionChannel id="k_slow" conductance="10pS" type="ionChannelHH" species="k
↪">
```

(continues on next page)

(continued from previous page)

```
...         <notes>K slow channel from Boyle and Cohen 2008</notes>
...         <gateHHtauInf id="n" instances="1">
...             <timeCourse type="fixedTimeCourse" tau="25.0007 ms"/>
...             <steadyState type="HHSigmoidVariable" rate="1" scale="15.8512 mV"␣
↪midpoint="19.8741 mV"/>
...         </gateHHtauInf>
...     </ionChannel>
... </neuroml>""")

>>> external_nml = NeuroMLDocument(ident='external_ex',
...     document_url='')
```

content
XML content for the document. Should be a complete NeuroML document rather than a fragment.

document_url
URL where the XML content of the document can be retrieved

class owmeta.neuroml.**NeuroMLProperty**(*resolver*, **args*, ***kwargs*)
Bases: `owmeta_core.dataobject_property.ObjectProperty`

Property for attaching NeuroML documents to resources

value_type
alias of *NeuroMLDocument*

owmeta.neuron module

owmeta.plot module

class owmeta.plot.**Plot**(*data=None*, **args*, ***kwargs*)
Bases: `owmeta_core.dataobject.DataObject`

Object for storing plot data in owmeta.

Parameters

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

Example usage ::

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

get_data()
Get the data stored for this plot.

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

owmeta.sources module

owmeta.translators module

owmeta.utils module

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

owmeta.website module

class `owmeta.website.`**`Website`**(*title=None*, ***kwargs*)

Bases: *`owmeta.document.BaseDocument`*

A representation of a website

`defined_augment`()

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

`identifier_augment`()

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

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

Raises

`IdentifierMissingException`

`title`

The official name for the website

`url`

A URL for the website

owmeta.worm module

owmeta.worm_common module

CHAPTER 2

For Users

2.1 owmeta Data Sources

The sources of data for owmeta are stored in the OpenWormData repository. A few `DataTranslators` translate these data into common owmeta data sources. You can list these by running:

```
owm source list
```

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

```
owm source show $SOURCE_IDENTIFIER
```

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

```
owm 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 owmeta Data

Below, each major element of the worm's anatomy that owmeta 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.

[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

- 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.

2.1.4 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.

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

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 (Resource Description Framework) 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 modeling

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).

Nervous system

For 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

C. elegans has very stable cell division patterns in the absence of mutations. This means that we can capture divisions in our database as static 'daughterOf' 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

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). With owmeta, supporting information can be attached to named graphs, which are groupings of statements with a URI attached to them. A named graph can have as many or as few statements as desired. Furthermore, a given triple can occur in multiple named graphs. 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 track responsibility for new uploads to the database through the OpenWormData Git repository. Each named graph is canonicalized – essentially, triples are sorted and written to a text file – and committed to a Git repository which gives us, at least, an email address and a timestamp for all modifications.

Access control

Data in owmeta are distributed as a bundle, a packaging structure which contains a set of canonicalized named graphs and, optionally, some files. Responsibility for restricting who can modify a bundle is, in the first instance, up to the bundle creator. When the bundle is actually distributed, the responsibility then falls on the distributor to ensure authentication of the bundle's provider and integrity of the bundle.

In OpenWorm, we create bundles from the OpenWormData GitHub repository. Access to the repository is managed by senior OpenWorm contributors. Bundles are deployed to Google Drive with write access controlled by Mark Watts. You can fetch OpenWorm bundles by adding a remote like this:

```
owm bundle remote add google-drive 'https://drive.google.com/uc?
↪id=1NYAcKdcvoFu5c7Nz3l4hK5UacG_eD56V&authuser=0&export=download'
```

`google-drive` can be substituted with any string.

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. Instead, we utilize bundle versioning to track revisions to the data. Each successive release of the bundle increments the bundle version number.

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.

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 owmeta, 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 owmeta. 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 owmeta.neuron import Neuron
>>> n = Neuron.query()
>>> n.receptor('UNC-13')
owmeta_core.statement.Statement(...obj=owmeta_core.dataobject_property.
↪ContextualizedPropertyValue(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 owmeta_core.context import Context
>>> from owmeta.worm import Worm
>>> from owmeta.network import Network
>>> ctx = Context('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
```

(continues on next page)

[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

(continued from previous page)

```
>>> w.neuron_network(nn)          # attach to the worm the neuron network
owmeta_core.statement.Statement(...)
>>> n = ctx(Neuron)('NeuronX')    # make a neuron
>>> n.receptor('UNC-13')          # state that the neuron has a UNC-13 type receptor
owmeta_core.statement.Statement(...)
>>> nn.neuron(n)                  # attach to the neuron network
owmeta_core.statement.Statement(...)
>>> ctx.save()            # 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 owmeta 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('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 owmeta, that kind of context-sensitivity is modeled by using `owmeta.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 owmeta, 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 owmeta_core.context import Context
>>> # from mymod import Widget   # my own OWM widget model
>>> # from bdw import Load # BigDataWarehouse API

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

>>> # 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(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://example.org/BDW/entities/Widget#12> <http...> <http://example.org/data/
↪imports/BDW_Widgets_2017-2018> .
<http://example.org/BDW/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 owmeta semantics, that when you query objects from the importing context, that the imported contexts will also be available to query.

2.4 Software Versioning

The owmeta library follows the semanitc versioning scheme. For the sake of versioning, the software interface consists of:

1. Extensions to the **owm** command line defined
2. All "public" definitions (i.e., those whose names do not begin with '_') in the *owmeta* package, sub-packages, and sub-modules
3. The format of RDF data generated by subclasses of `owmeta_core.dataobject.DataObject` and defined in the *owmeta* package, sub-packages, and sub-modules
4. The API documentation for the *owmeta* package, sub-packages, and sub-modules

In addition, any changes to the packages released on PyPI mandates at least a patch version increment.

For Git, our software version control system, software releases will be represented as tags in the form `v$semantic_version` with all components of the semantic version represented.

2.4.1 Documentation versioning

The documentation will have a distinct version number from the software. The version numbers for the documentation must change at least as often as the software versioning since the relationship of the documentation to the software necessarily changes. However, changes _only_ to the non-API documentation will not be a cause for a change to any of the components of the software version number. For documentation releases which coincide with software releases, the documentation version number will simply be the software version number. Any subsequent change to documentation between software releases will compel an increase in the documentation version number by one. The documentation version number for such documentation releases will be represented as `${software_version}+docs${documentation_increment}`.

2.5 Python Release Compatibility

All Python releases will be supported until they reach their official end-of-life, typically reported as "Release Schedule" PEPs (search "release schedule" on the PEP index) Thereafter, any regressions due to dependencies of owmeta dropping support for an EOL Python version, or due to a change in owmeta making use of a feature in a still-supported Python release will only be fixed for the sake of OpenWorm projects when requested by an issue on our tracker or for other projects when a compelling case can be made.

This policy is intended to provide support to most well-maintained projects which depend on owmeta while not over-burdening developers.

CHAPTER 3

For Developers

3.1 Testing in owmeta

3.1.1 Preparing for tests

Within the owmeta project directory, owmeta can be installed for development and testing like this:

```
pip install --editable .
```

The project database should be populated like:

```
owm 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. All marks are listed in pytest.ini under ‘markers’.

3.2 Adding documentation

Documentation for owmeta 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 `owmeta/__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 owmeta coding standards

Pull requests are *required* to follow the PEP-8 Guidelines for contributions of Python code to owmeta, 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

o

owmeta, 3
owmeta.bibtex, 6
owmeta.bibtex_customizations, 6
owmeta.biology, 6
owmeta.cell_common, 6
owmeta.channel_common, 6
owmeta.cli_hints, 7
owmeta.commands, 4
owmeta.data_trans, 4
owmeta.data_trans.bibtex, 4
owmeta.data_trans.common_data, 5
owmeta.data_trans.context_merge, 5
owmeta.data_trans.data_with_evidence_ds, 5
owmeta.document, 7
owmeta.documentContext, 8
owmeta.evidence, 8
owmeta.experiment, 10
owmeta.neuroml, 10
owmeta.plot, 11
owmeta.utils, 12
owmeta.website, 12
owmeta.worm_common, 12

Index

A

author (*owmeta.document.Document attribute*), 8
author() (*in module owmeta.bibtex_customizations*), 6

B

BaseDocument (*class in owmeta.document*), 7
bibtex_to_document() (*in module owmeta.bibtex*), 6
BibTexDataSource (*class in owmeta.data_trans.bibtex*), 4
BibTexDataTranslator (*class in owmeta.data_trans.bibtex*), 4
BiologyType (*class in owmeta.biology*), 6

C

CHANNEL_RDF_TYPE (*in module owmeta.channel_common*), 6
Configurable (*class in owmeta*), 3
content (*owmeta.neuroml.NeuroMLDocument attribute*), 11
ContextMergeDataTranslator (*class in owmeta.data_trans.context_merge*), 5
customizations() (*in module owmeta.bibtex_customizations*), 6

D

DataWithEvidenceDataSource (*class in owmeta.data_trans.data_with_evidence_ds*), 5
date (*owmeta.document.Document attribute*), 8
defined_augment() (*owmeta.document.Document method*), 7
defined_augment() (*owmeta.evidence.Evidence method*), 9
defined_augment() (*owmeta.website.Website method*), 12
Document (*class in owmeta.document*), 7
document_url (*owmeta.neuroml.NeuroMLDocument attribute*), 11
DocumentContext (*class in owmeta.documentContext*), 8
DocumentContextMeta (*class in owmeta.documentContext*), 8
doi (*owmeta.document.Document attribute*), 8
doi() (*in module owmeta.bibtex_customizations*), 6

E

Evidence (*class in owmeta.evidence*), 8
evidence_for() (*in module owmeta.evidence*), 9
EvidenceDataSource (*class in owmeta.data_trans.bibtex*), 5
EvidenceError, 8
Experiment (*class in owmeta.experiment*), 10

G

get() (*owmeta.Configurable method*), 3
get_conditions() (*owmeta.experiment.Experiment method*), 10
get_data() (*owmeta.plot.Plot method*), 11

I

identifier_augment() (*owmeta.document.Document method*), 7
identifier_augment() (*owmeta.evidence.Evidence method*), 9
identifier_augment() (*owmeta.website.Website method*), 12
input_type (*owmeta.data_trans.bibtex.BibTexDataTranslator attribute*), 4

N

NeuroMLDocument (*class in owmeta.neuroml*), 10
NeuroMLProperty (*class in owmeta.neuroml*), 11

O

output_type (*owmeta.data_trans.bibtex.BibTexDataTranslator attribute*), 4

output_type (*owmeta.data_trans.context_merge.ContextMergeDataTranslator attribute*), 5
owmeta (*module*), 3
owmeta.bibtex (*module*), 6
owmeta.bibtex_customizations (*module*), 6
owmeta.biology (*module*), 6
owmeta.cell_common (*module*), 6
owmeta.channel_common (*module*), 6
owmeta.cli_hints (*module*), 7
owmeta.commands (*module*), 4
owmeta.data_trans (*module*), 4
owmeta.data_trans.bibtex (*module*), 4
owmeta.data_trans.common_data (*module*), 5
owmeta.data_trans.context_merge (*module*), 5
owmeta.data_trans.data_with_evidence_ds (*module*), 5
owmeta.document (*module*), 7
owmeta.documentContext (*module*), 8
owmeta.evidence (*module*), 8
owmeta.experiment (*module*), 10
owmeta.neuroml (*module*), 10
owmeta.plot (*module*), 11
owmeta.utils (*module*), 12
owmeta.website (*module*), 12
owmeta.worm_common (*module*), 12

P

Plot (*class in owmeta.plot*), 11
pmid (*owmeta.document.Document attribute*), 8
PubmedRetrievalException, 7

Q

query_context() (*in module owmeta.evidence*), 10

R

reference (*owmeta.evidence.Evidence attribute*), 9
reference (*owmeta.experiment.Experiment attribute*), 10
refutes (*owmeta.evidence.Evidence attribute*), 9

S

set_data() (*owmeta.plot.Plot method*), 11
supports (*owmeta.evidence.Evidence attribute*), 9

T

title (*owmeta.document.Document attribute*), 8
title (*owmeta.website.Website attribute*), 12
translate() (*owmeta.data_trans.bibtex.BibTexDataTranslator method*), 5
translate() (*owmeta.data_trans.context_merge.ContextMergeDataTranslator method*), 5

U

update_from_wormbase() (*owmeta.document.Document method*), 8
uri (*owmeta.document.Document attribute*), 8
url (*owmeta.website.Website attribute*), 12

V

value_type (*owmeta.neuroml.NeuroMLProperty attribute*), 11

W

wbid (*owmeta.document.Document attribute*), 8
Website (*class in owmeta.website*), 12
wormbaseid (*owmeta.document.Document attribute*), 8
WormbaseRetrievalException, 7

Y

year (*owmeta.document.Document attribute*), 8