Object model documentation

Release 0.0.6

Jonathan Karr, Arthur Goldberg

Jun 18, 2019

Contents

obj_model allows developers to define standalone (i.e. separate from databases) schemas using a syntax similar to Django. The *obj_model.io* module provides methods to serialize and deserialize schema objects to/from Excel, csv, and tsv file(s).

CHAPTER 1

Contents

1.1 Installation

1.1.1 Prerequisites

- Python
- Pip

1.1.2 Latest release From PyPI

Run the following command to install the latest release from PyPI:

```
pip install obj_model
```

1.1.3 Latest revision from GitHub

Run the following command to install the latest version from GitHub:

```
pip install git+https://github.com/KarrLab/obj_model.git#egg=obj_model
```

1.2 Overview

obj_model allows developers to define standalone (i.e. separate from databases) schemas using a syntax similar to Django. The *obj_model.io* module provides methods to serialize and deserialize schema objects to/from Excel, csv, and tsv file(s).

1.2.1 Defining schemas

Each schema is composed of one or models (subclasses of `Model`) each of which has one or more attributes (instances of `Attribute` and its subclasses). The following shows an example of a schema for a lab member:

```
class Member(Model):
    first_name = StringAttribute()
    last_name = StringAttribute()
```

Multiple attributes types are provided:

- `BooleanAttribute`
- `EnumAttribute`
- `IntegerAttribute`, `PositiveIntegerAttribute`
- `FloatAttribute`
- `StringAttribute`, `LongStringAttribute`, `RegexAttribute`, `UrlAttribute`, `SlugAttribute`
- `DateAttribute`, `TimeAttribute`, `DateTimeAttribute`

Four related attribute types (`OneToOneAttribute`, `OneToManyAttribute`, `ManyToOneAttribute`, and `ManyToManyAttribute`) are provided to enable relationships among objects. Each constructor includes an optional argument *related_name* which when provided automatically constructs a reverse attribute between the instances:

```
class Lab(Model):
    name = StringAttribute()
    url = UrlAttribute()

class Member(Model):
    first_name = StringAttribute()
    last_name = StringAttribute()
    lab = ManyToOneAttribute(Lab, related_name='members')
```

Do not choose attribute names that would clash with with built-in attributes or methods of classes, such as *validate*, *serialize*, and *deserialize*.

1.2.2 Instantiating objects

The module automatically adds optional keyword arguments to the constructor for each type. Thus objects can be constructed as illustrated below:

```
lab = Lab(name='Karr Lab')
member = Member(first_name='Jonathan', last_name='Karr', lab=lab)
```

1.2.3 Getting and setting object attributes

Objects attributes can be get and set as shown below:

```
name = lab.name
lab.url = 'http://www.karrlab.org'
```

Related attributes can also be edited as shown below:

```
new_member = Member(first_name='new', last_name='guy')
lab.members = [new_member]
```

-to-many and many-to- attribute and related attribute values are instances of `RelatedManager` which is a subclass of `set`. Thus, their values can also be edited with set methods such as *add*, *clear*, *remove*, and *update*. `RelatedManager` provides three additional methods:

- *create*: *object.related_objects.create(**kwargs)* is syntatic sugar for *object.attribute.add(RelatedObject(**kwargs))*
- *get_one*: this returns a related object with attribute values equal to the supplies keyward argments
- *get*: this returns the subset of the related objects with attribute values equal to the supplied keyword argments

1.2.4 Meta information

To allow developers to customize the behavior of each `Model` subclass, `Model` provides an internal *Meta* class (`Model.Meta`). This provides several attributes:

- *attribute_order*: `tuple` of attribute names; controls order in which attributes should be printed when serialized
- *frozen_columns*: `int`: controls how many columns should be frozen when the model is serialized to Excel
- *ordering*: `tuple` of attribute names; controls the order in which objects should be printed when serialized
- *tabular_orientation*: `TabularOrientation`: controls orientation (row, column, inline) of model when serialized
- *unique_together*: `tuple` of attribute names; controls what tuples of attribute values must be unique
- *verbose_name*: verbose name of the model; used for (de)serialization
- *verbose_name_plural*: plural verbose name of the model; used for (de)serialization

1.2.5 Validation

To facilitate data validation, the module allows developers to specify how objects should be validated at several levels:

- Attribute: `Attribute` defines a method *validate* which can be used to validate individual attribute values. Attributes of (e.g. *min*, *max*, *min_length*, *max_length*, etc.) these classes can be used to customize this validation
- Object: `Model` defines a method *validate* which can be used to validate entire object instances
- Model: `Model` defines a class method *validate_unique* which can be used to validate sets of object instances of the same type. This is customized by setting (a) the *unique* attribute of each model type's attrbutes or (b) the *unique_together* attribute of the model's *Meta* class.
- Dataset: `Validator` can be subclasses provide additional custom validation of entire datasets

Validation does not occur automatically, rather users must call validate() when it is needed.

1.2.6 Equality, differencing

To facilitate comparison between objects, the `Model` provides two methods

- *is_equal*: returns `True` if two `Model` instances are semantically equal (all attribute values are recursively equal)
- *difference*: returns a textual description of the difference(s) between two objects

1.2.7 Serialization/deserialization

The *io* module provides methods to serialize and deserialize schema objects to/from Excel, csv, and tsv files(s). `Model.Meta` provides several attributes to enable developers to control how each model is serialized. Please see the "Meta information" section above for more information.

1.2.8 Utilities

The *utils* module provides several additional utilities for manipulating `Model` instances.

1.3 About

1.3.1 License

The software is released under the MIT license

1.3.2 Development team

This package was developed by the Karr Lab at the Icahn School of Medicine at Mount Sinai in New York, USA.

- Jonathan Karr
- Arthur Goldberg

1.3.3 Acknowledgements

This work was supported by a National Science Foundation INSPIRE award [grant number 1649014].

1.3.4 Questions and comments

Please contact the Karr Lab with any questions or comments.