Alchemize Documentation

Release 0.13.2

John Vrbanac

Oct 15, 2018

Contents

Alchemize is designed to be a simple serialization and deserialization library. The primary use-case for Alchemize is to allow for users to quickly build ReST clients using simple model mappings to transform data from Python objects to a serializable form and vice-versa.

The power of Alchemize is that you can use it to augment existing model structures from other libraries. For example, you can use Alchemize to easily serialize your ORM models.

CHAPTER 1

Installation

Alchemize is available on PyPI

```
pip install alchemize
```

1.1 Using Alchemize

Alchemize relies on a explicit mapping structure that is defined on your model class. The mapping combined with the appropriate class mix-in will enable your object to serialize and deserialize to a supported format by Alchemize.

1.1.1 Simple Example

In the following example, we'll be taking a portion of the sample response from the GitHub's API to retrieve data on a single user.

Input JSON

```
{
    "login": "octocat",
    "id": 1,
    "avatar_url": "https://github.com/images/error/octocat_happy.gif",
    "gravatar_id": "",
    "url": "https://api.github.com/users/octocat",
    "html_url": "https://github.com/octocat",
    "type": "User",
    "site_admin": false,
    "name": "monalisa octocat",
    "company": "GitHub",
    "blog": "https://github.com/blog",
    "location": "San Francisco",
    "email": "octocat@github.com",
    "hireable": false,
```

(continues on next page)

(continued from previous page)

```
    "bio": "There once was...",
    "public_repos": 2,
    "public_gists": 1,
    "followers": 20,
    "following": 0,
    "created_at": "2008-01-14T04:33:35Z",
    "updated_at": "2008-01-14T04:33:35Z"
}
```

Example Mapping

```
from alchemize import JsonMappedModel, Attr

class ExampleClass(JsonMappedModel):
    __mapping__ = {
        'id': Attr('user_id', int),
        'login': Attr('login', str),
        "email": Attr('email', str),
        "name": Attr('name' , str)
    }
    ...
```

Now that we have some sample JSON and a mapping on a class, we can put this information through the JsonTransmuter to get two-way conversion of our data.

Example Transmutation

```
from alchemize import JsonTransmuter

# From JSON to our Python Mapped Model
result_model = JsonTransmuter.transmute_from(json_str, ExampleClass)

# From our Python Mapped Model back to JSON
result_json = JsonTransmuter.transmute_to(result_model)
```

If you look at the resulting model and JSON string, you might notice that the only information that is carried over is what has been explicitly mapped.

1.1.2 Nested Mapped Models

Alchemize supports the ability to de/serialize child mapped models as well. This allows for you to explicitly map your data into nested Python objects and let Alchemize handle the serialization and deserialization of the entire data structure at once.

Example:

We want to convert the following JSON into appropriate Python objects.

```
{
    "id": 12345,
    "users": [
        {
            "name": "Foster Person",
            "email": "foster.person@example.com"
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "name": "Other Person",
            "email": "other.person@example.com"
        }
    ]
}
```

```
from alchemize import JsonMappedModel, Attr

class User(JsonMappedModel):
    __mapping__ = {
        'name': Attr('name', str),
        'email': Attr('email', str)
    }

class Project(JsonMappedModel):
    __mapping__ = {
        'id': Attr('project_id', int),
        'users': Attr('users', [User])
    }
```

We can now deserialize the data into our models using the JsonTransmuter

```
from alchemize import JsonTransmuter

result_model = JsonTransmuter.transmute_from(json_str, Project)

result_model.users[0].name # 'Foster Person'
result_model.users[1].name # 'Other Person'
```

We have successfully converted our JSON structure into a easily usable Python object structure.

For more information on how to define your mappings, take a look at the *Alchemize API Documentation*

1.1.3 Excluding Attributes for Serialization

For specific data models there are instances where you don't want to serialize certain attributes. For example, you're pulling user information from a database but you don't want to serialize the password hash or some other internal value. This is done by setting the `serialize=False` argument on your `Attr`.

```
from alchemize import JsonMappedModel, Attr

class User(JsonMappedModel):
    __mapping__ = {
        'name': Attr('name', str),
        'email': Attr('email', str),
        'password': Attr('password', str, serialize=False),
    }
```

Note: The `serialize` setting can be overridden by the transmuter if explicited set during the `transmute_to(...)` call.

1.1.4 Wrapped Objects

Some API responses can wrap the information you're interested in inside a container object. There are many reasons for this, but often from a model interface perspective, you just want to represent the data itself and not the container. To handle this use-case, Alchemize provides the `__wrapped_attr_name__` option.

This option allows for parsing this JSON into the following single model

```
{
    "#item": {
        "name": "John Doe",
        "email": "rando@doe.com",
    }
}
```

```
from alchemize import JsonMappedModel, Attr

class User(JsonMappedModel):
    __wrapped_attr_name__ = '#item'
    __mapping__ = {
        'name': Attr('name', str),
        'email': Attr('email', str),
    }
```

1.1.5 Simple Helper Usage

If you don't really care about a custom serialization implementation or hook-in you can use the basic helper models which handle will transmute to/from on their own.

```
from alchemize import JsonModel, Attr


class User(JsonModel):
    __mapping__ = {
        'name': Attr('name', str),
        'email': Attr('email', str),
        'password': Attr('password', str, serialize=False),
    }


model = User.from_dict({
    'name': 'thing',
    'email': 'thing@thing.corp',
    'password': 'my-password',
})

json_dict = model.as_dict()

# You can also set attributes on instance creation
model = User(name='thing', email='thing@thing.corp')
```

1.2 Alchemize API Documentation

1.2.1 Transmuters

class alchemize.**JsonTransmuter**

classmethod transmute_from(*data*, *mapped_model_type*, *coerce_values=False*, *decoder=None*, *decoder_kwargs=None*)

Converts a JSON string or dict into a corresponding Mapping Object.

Parameters

- **data** – JSON data in string or dictionary form.
- **mapped_model_type** – A type that extends the JsonMappedModel base.
- **coerce_values** – Boolean value to allow for values with python types to be coerced with their mapped type.
- **decoder** – A module that implements loads(...).
- **decoder_kwargs** – A dictionary containing kwargs to use with the decoder.

Returns An instance of your mapped model type.

classmethod transmute_to(*mapped_model*, *to_string=True*, *assign_all=False*, *coerce_values=True*, *serialize_all=False*, *encoder=None*, *encoder_kwargs=None*)

Converts a model based off of a JsonMappedModel into JSON.

Parameters

- **mapped_model** – An instance of a subclass of JsonMappedModel.
- **to_string** – Boolean value to disable the return of a string and return a dictionary instead.
- **assign_all** – Boolean value to force assignment of all values, including null values.
- **coerce_values** – Boolean value to allow for values with python types to be coerced with their mapped type.
- **serialize_all** – Boolean value that allows for you to force serialization of values regardless of the attribute settings.
- **encoder** – module that implements dumps(...).
- **encoder_kwargs** – A dictionary containing kwargs to be used with the encoder.

Returns A string or dictionary containing the JSON form of your mapped model.

class alchemize.**AbstractBaseTransmuter**

The abtract base class from which all Transmuters are built.

classmethod transmute_from(*data*, *mapped_model_type*)

Generic Abstract Base Method to deserialize into a Python object

Parameters **mapped_model_type** – A type that extends BaseMappedModel.

Returns The an instance of the passed in mapped model type containing the deserialized data.

classmethod transmute_to(*mapped_model*)

Generic Abstract Base Method to convert to serialized form

Parameters **mapped_model** – An instance of a class based from BaseMappedModel.

Returns A serialized or serializable form of your mapped model.

1.2.2 Mapped Models

class alchemize.**Attr**(*attr_name*, *attr_type*, *serialize=True*, *required=False*, *coerce=None*)
Attribute Definition

Parameters

- **name** – Python attribute name
- **type** – Attribute type (e.g str, int, dict, etc)
- **serialize** – Determines if the attribute can be serialized
- **required** – Forces attribute to be defined
- **coerce** – Forces attribute to be coerced to its type (primitive types)

class alchemize.**JsonMappedModel**
Creates an explicit mapping for de/serialization by the JsonTransmuter

Map Structure:

```
'json_attr_name': Attr('python_attr_name', StorageType)
```

Mapping Types:

```
__mapping__ = {
    'name': Attr('name', str),
    'number': Attr('number', int),
    'dict': Attr('sample_dict', dict),
    'list': Attr('sample_list', list),
    'child': Attr('child', ChildModel),
    'children': Attr('children', [ChildModel])
}
```

alchemize.mapping.**get_key_paths**(*model*, *sep='/'*, *prefix=''*)
Walks a model class and returns a list of all key paths

Parameters

- **model** – Mapped Model instance or class
- **sep** – Separator used to join the keys together
- **prefix** – Prefix to add to all keys

Returns List of key paths

alchemize.mapping.**get_normalized_map**(*model*)
Normalizes mapping data to support backward compatibility.

1.2.3 Helpers

class alchemize.**JsonModel**(**attrs*)
Model helper that is designed to provide common usage methods.

as_dict(*serialize_all=False*)
Converts the model into a dictionary.

as_json(*serialize_all=False*)
Converts the model into a JSON string.

classmethod from_dict(*data*, ***transmute_options*)
Creates a new instance of the model from a dictionary.

classmethod from_json(*data*, ***transmute_options*)
Creates a new instance of the model from a JSON string.

update(***attrs*)
Updates object attributes with specified kwarg values.

class alchemize.**JsonListModel**(***attrs*)
The list model helper is designed to be a top-level container for more complex list structures.

While you can map any number of attributes to this model, it expects a special mapping to the 'collection' attribute that it'll alias normal iterable operations to.

Mapping Usage:

```
'my-items': Attr('collection', [ChildModel])
```

Note: The child items should be mapped into the 'collection' attribute to properly use this helper.

append(*item*)
Appends an item to the collection.

as_dict(*serialize_all=False*)
Converts the model into a dictionary.

as_json(*serialize_all=False*)
Converts the model into a JSON string.

extend(*items*)
Appends multiple items to the collection.

classmethod from_dict(*data*, ***transmute_options*)
Creates a new instance of the model from a dictionary.

classmethod from_json(*data*, ***transmute_options*)
Creates a new instance of the model from a JSON string.

update(***attrs*)
Updates object attributes with specified kwarg values.

1.2.4 Exceptions

class alchemize.**AlchemizeError**
Base Exception for all Alchemize errors.

class alchemize.transmute.**RequiredAttributeError**(*attribute_name*)
Exception that is raised when attempting to retrieve/apply an attribute that isn't available.

class alchemize.transmute.**UnsupportedMappedModelError**
Exception that is raised when attempting to transmute a model that is not supported by the specified transmuter.

1.2.5 Custom Types

class alchemize.**ExpandedType**
Custom Expanded Type Definition (Unstable Feature)

classmethod deserialize(*attr_type*, *value*)
Deserialization implementation for the type.

classmethod serialize(*value*)
Serialization implementation for the type.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index