marshmallow-jsonapi

Release 0.20.0

Jun 18, 2018

Contents

1 Installation **3**

2 Guide **5**
2.1 Quickstart 5

3 API Reference **15**
3.1 API Reference 15

4 Project info **21**
4.1 Changelog 21
4.2 Authors 26
4.3 Contributing Guidelines 27
4.4 License 28

5 Links **31**

Python Module Index **33**

Release v0.20.0. (*Changelog*)

JSON API 1.0 (https://jsonapi.org) formatting with marshmallow.

marshmallow-jsonapi provides a simple way to produce JSON API-compliant data in any Python web framework.

```
from marshmallow_jsonapi import Schema, fields

class PostSchema(Schema):
    id = fields.Str(dump_only=True)
    title = fields.Str()

    author = fields.Relationship(
        related_url='/authors/{author_id}',
        related_url_kwargs={'author_id': '<author.id>'},
    )

    comments = fields.Relationship(
        related_url='/posts/{post_id}/comments',
        related_url_kwargs={'post_id': '<id>'},
        # Include resource linkage
        many=True, include_resource_linkage=True,
        type_='comments'
    )

    class Meta:
        type_ = 'posts'
        strict = True

post_schema = PostSchema()
post_schema.dump(post).data
# {
#     "data": {
#         "id": "1",
#         "type": "posts"
#         "attributes": {
#             "title": "JSON API paints my bikeshed!"
#         },
#         "relationships": {
#             "author": {
#                 "links": {
#                     "related": "/authors/9"
#                 }
#             },
#             "comments": {
#                 "data": [
#                     {"id": 5, "type": "comments"},
#                     {"id": 12, "type": "comments"}
#                 ],
#                 "links": {
#                     "related": "/posts/1/comments/"
#                 }
#             }
#         },
#     }
# }
```

CHAPTER 1

Installation

```
pip install marshmallow-jsonapi
```

CHAPTER 2

Guide

2.1 Quickstart

Note: The following guide assumes some familiarity with the marshmallow API. To learn more about marshmallow, see its official documentation at https://marshmallow.readthedocs.io.

2.1.1 Declaring schemas

Let's start with a basic post "model".

```
class Post:
    def __init__(self, id, title):
        self.id = id
        self.title = title
```

Declare your schemas as you would with marshmallow.

A *Schema* **MUST** define:

- An `id` field
- The `type_` class Meta option

It is **RECOMMENDED** to set strict mode to `True`.

Automatic self-linking is supported through these Meta options:

- `self_url` specifies the URL to the resource itself
- `self_url_kwargs` specifies replacement fields for `self_url`
- `self_url_many` specifies the URL the resource when a collection (many) are serialized

```
from marshmallow_jsonapi import Schema, fields

class PostSchema(Schema):
    id = fields.Str(dump_only=True)
    title = fields.Str()

    class Meta:
        type_ = 'posts'
        self_url = '/posts/{id}'
        self_url_kwargs = {'id': '<id>'}
        self_url_many = '/posts/'
        strict = True
```

These URLs can be auto-generated by specifying `self_view`, `self_view_kwargs` and `self_view_many` instead when using the *Flask integration*.

2.1.2 Serialization

Objects will be serialized to JSON API documents with primary data.

```
post = Post(id='1', title='Django is Omakase')
PostSchema().dump(post)
# {
#     'data': {
#         'id': '1',
#         'type': 'posts',
#         'attributes': {'title': 'Django is Omakase'},
#         'links': {'self': '/posts/1'}
#     },
#     'links': {'self': '/posts/1'}
# }
```

2.1.3 Relationships

The `Relationship` field is used to serialize relationship objects. For example, a Post may have an author and comments associated with it.

```
class User:
    def __init__(self, id, name):
        self.id = id
        self.name = name

class Comment:
    def __init__(self, id, body, author):
        self.id = id
        self.body = body
        self.author = author

class Post:
    def __init__(self, id, title, author, comments=None):
        self.id = id
        self.title = title
        self.author = author    # User object
        self.comments = [] if comments is None else comments    # Comment objects
```

To serialize links, pass a URL format string and a dictionary of keyword arguments. String arguments enclosed in < > will be interpreted as attributes to pull from the object being serialized. The relationship links can automatically be generated from Flask view names when using the *Flask integration*.

```
class PostSchema(Schema):
    id = fields.Str(dump_only=True)
    title = fields.Str()

    author = fields.Relationship(
        self_url='/posts/{post_id}/relationships/author',
        self_url_kwargs={'post_id': '<id>'},
        related_url='/authors/{author_id}',
        related_url_kwargs={'author_id': '<author.id>'}
    )

    class Meta:
        type_ = 'posts'
        strict = True

user = User(id='94', name='Laura')
post = Post(id='1', title='Django is Omakase', author=user)
PostSchema().dump(post)
# {
#     'data': {
#         'id': '1',
#         'type': 'posts',
#         'attributes': {'title': 'Django is Omakase'},
#         'relationships': {
#             'author': {
#                 'links': {
#                     'self': '/posts/1/relationships/author',
#                     'related': '/authors/94'
#                 }
#             }
#         }
#     }
# }
```

Resource linkages

You can serialize resource linkages by passing `include_resource_linkage=True` and the resource `type_` argument.

```
class PostSchema(Schema):
    id = fields.Str(dump_only=True)
    title = fields.Str()

    author = fields.Relationship(
        self_url='/posts/{post_id}/relationships/author',
        self_url_kwargs={'post_id': '<id>'},
        related_url='/authors/{author_id}',
        related_url_kwargs={'author_id': '<author.id>'},
        # Include resource linkage
        include_resource_linkage=True,
        type_='users'
    )
```

(continues on next page)

(continued from previous page)

```
    class Meta:
        type_ = 'posts'
        strict = True

PostSchema().dump(post)
# {
#     'data': {
#         'id': '1',
#         'type': 'posts',
#         'attributes': {'title': 'Django is Omakase'},
#         'relationships': {
#             'author': {
#                 'data': {'type': 'users', 'id': '94'},
#                 'links': {
#                     'self': '/posts/1/relationships/author',
#                     'related': '/authors/94'
#                 }
#             }
#         }
#     }
# }
```

Compound documents

Compound documents allow to include related resources into the request with the primary resource. In order to include objects, you have to define a *Schema* for the respective relationship, which will be used to render those objects.

```
class PostSchema(Schema):
    id = fields.Str(dump_only=True)
    title = fields.Str()

    comments = fields.Relationship(
        related_url='/posts/{post_id}/comments',
        related_url_kwargs={'post_id': '<id>'},
        many=True, include_resource_linkage=True,
        type_='comments',
        # define a schema for rendering included data
        schema='CommentSchema'
    )

    author = fields.Relationship(
        self_url='/posts/{post_id}/relationships/author',
        self_url_kwargs={'post_id': '<id>'},
        related_url='/authors/{author_id}',
        related_url_kwargs={'author_id': '<author.id>'},
        include_resource_linkage=True,
        type_='users'
    )

    class Meta:
        type_ = 'posts'
        strict = True

class CommentSchema(Schema):
```

(continues on next page)

(continued from previous page)

```
    id = fields.Str(dump_only=True)
    body = fields.Str()

    author = fields.Relationship(
        self_url='/comments/{comment_id}/relationships/author',
        self_url_kwargs={'comment_id': '<id>'},
        related_url='/comments/{author_id}',
        related_url_kwargs={'author_id': '<author.id>'},
        type_='users',
        # define a schema for rendering included data
        schema='UserSchema',
    )

    class Meta:
        type_ = 'comments'
        strict = True

class UserSchema(Schema):
    id = fields.Str(dump_only=True)
    name = fields.Str()

    class Meta:
        type_ = 'users'
        strict = True
```

Just as with nested fields the `schema` can be a class or a string with a simple or fully qualified class name. Make sure to import the schema beforehand.

Now you can include some data in a dump by specifying the `include_data` argument (also supports nested relations via the dot syntax).

```
armin = User(id='101', name='Armin')
laura = User(id='94', name='Laura')
steven = User(id='23', name='Steven')
comments = [Comment(id='5', body='Marshmallow is sweet like sugar!', author=steven),
            Comment(id='12', body='Flask is Fun!', author=armin)]
post = Post(id='1', title='Django is Omakase', author=laura, comments=comments)

PostSchema(include_data=('comments', 'comments.author')).dump(post)
# {
#     'data': {
#         'id': '1',
#         'type': 'posts',
#         'attributes': {'title': 'Django is Omakase'},
#         'relationships': {
#             'author': {
#                 'data': {'type': 'users', 'id': '94'},
#                 'links': {
#                     'self': '/posts/1/relationships/author',
#                     'related': '/authors/94'
#                 }
#             },
#             'comments': {
#                 'data': [
#                     {'type': 'comments', 'id': '5'},
#                     {'type': 'comments', 'id': '12'}
#                 ],
```

(continues on next page)

(continued from previous page)

```
#                   'links': {
#                       'related': '/posts/1/comments'
#                   }
#               }
#           }
#       },
#       'included': [
#           {
#               'id': '5',
#               'type': 'comments',
#               'attributes': {'body': 'Marshmallow is sweet like sugar!'},
#               'relationships': {
#                   'author': {
#                       'data': {'type': 'users', 'id': '23'},
#                       'links': {
#                           'self': '/comments/5/relationships/author',
#                           'related': '/comments/23'
#                       }
#                   }
#               }
#           },
#           {
#               'id': '12',
#               'type': 'comments',
#               'attributes': {'body': 'Flask is Fun!'},
#               'relationships': {
#                   'author': {
#                       'data': {'type': 'users', 'id': '101'},
#                       'links': {
#                           'self': '/comments/12/relationships/author',
#                           'related': '/comments/101'
#                       }
#                   }
#               },
#
#           },
#           {
#               'id': '23',
#               'type': 'users',
#               'attributes': {'name': 'Steven'}
#           },
#           {
#               'id': '101',
#               'type': 'users',
#               'attributes': {'name': 'Armin'}
#           }
#       ]
# }
```

2.1.4 Meta Information

The `DocumentMeta` field is used to serialize the meta object within a document's "top level".

```
from marshmallow_jsonapi import Schema, fields
```

(continues on next page)

(continued from previous page)

```
class UserSchema(Schema):
    id = fields.Str(dump_only=True)
    name = fields.Str()
    document_meta = fields.DocumentMeta()

    class Meta:
        type_ = 'users'
        strict = True

user = {'name': 'Alice', 'document_meta': {'page': {'offset': 10}}}
UserSchema().dump(user)
# {
#     "meta": {
#         "page": {
#             "offset": 10
#         }
#     },
#     "data": {
#         "id": "1",
#         "type": "users"
#         "attributes": {"name": "Alice"},
#     }
# }
```

The *ResourceMeta* field is used to serialize the meta object within a resource object.

```
from marshmallow_jsonapi import Schema, fields

class UserSchema(Schema):
    id = fields.Str(dump_only=True)
    name = fields.Str()
    resource_meta = fields.ResourceMeta()

    class Meta:
        type_ = 'users'
        strict = True

user = {'name': 'Alice', 'resource_meta': {'active': True}}
UserSchema().dump(user)
# {
#     "data": {
#         "type": "users",
#         "attributes": {"name": "Alice"},
#         "meta": {
#             "active": true
#         }
#     }
# }
```

2.1.5 Errors

`Schema.load()` and `Schema.validate()` will return JSON API-formatted Error objects.

```
from marshmallow_jsonapi import Schema, fields
from marshmallow import validate, ValidationError
```

(continues on next page)

(continued from previous page)

```
class AuthorSchema(Schema):
    id = fields.Str(dump_only=True)
    first_name = fields.Str(required=True)
    last_name = fields.Str(required=True)
    password = fields.Str(load_only=True, validate=validate.Length(6))
    twitter = fields.Str()

    class Meta:
        type_ = 'authors'
        strict = True

author_data = {
    'data': {
        'type': 'users',
        'attributes': {
            'first_name': 'Dan',
            'password': 'short'
        }
    }
}
AuthorSchema().validate(author_data)
# {
#     'errors': [
#         {
#             'detail': 'Missing data for required field.',
#             'source': {
#                 'pointer': '/data/attributes/last_name'
#             }
#         },
#         {
#             'detail': 'Shorter than minimum length 6.',
#             'source': {
#                 'pointer': '/data/attributes/password'
#             }
#         }
#     ]
# }
```

If an invalid "type" is passed in the input data, an *IncorrectTypeError* is raised.

```
from marshmallow_jsonapi.exceptions import IncorrectTypeError

author_data = {
    'data': {
        'type': 'invalid-type',
        'attributes': {
            'first_name': 'Dan',
            'last_name': 'Gebhardt',
            'password': 'verysecure'
        }
    }
}

try:
    AuthorSchema().validate(author_data)
except IncorrectTypeError as err:
```

(continues on next page)

(continued from previous page)

```
    pprint(err.messages)
# {
#     'errors': [
#         {
#             'detail': 'Invalid type. Expected "users".',
#             'source': {
#                 'pointer': '/data/type'
#             }
#         }
#     ]
# }
```

2.1.6 Inflection

You can optionally specify a function to transform attribute names. For example, you may decide to follow JSON API's recommendation to use "dasherized" names.

```
from marshmallow_jsonapi import Schema, fields

def dasherize(text):
    return text.replace('_', '-')

class UserSchema(Schema):
    id = fields.Str(dump_only=True)
    first_name = fields.Str(required=True)
    last_name = fields.Str(required=True)

    class Meta:
        type_ = 'users'
        inflect = dasherize

UserSchema().dump(user)
# {
#     'data': {
#         'id': '9',
#         'type': 'users',
#         'attributes': {
#             'first-name': 'Dan',
#             'last-name': 'Gebhardt'
#         }
#     }
# }
```

2.1.7 Flask integration

marshmallow-jsonapi includes optional utilities to integrate with Flask.

A Flask-specific schema in *marshmallow_jsonapi.flask* can be used to auto-generate self-links based on view names instead of hard-coding URLs.

Additionally, the `Relationship` field in the *marshmallow_jsonapi.flask* module allows you to pass view names instead of path templates to generate relationship links.

```
from marshmallow_jsonapi import fields
from marshmallow_jsonapi.flask import Relationship, Schema

class PostSchema(Schema):
    id = fields.Str(dump_only=True)
    title = fields.Str()

    author = fields.Relationship(
        self_view='post_author',
        self_url_kwargs={'post_id': '<id>'},
        related_view='author_detail',
        related_view_kwargs={'author_id': '<author.id>'}
    )

    comments = Relationship(
        related_view='post_comments',
        related_view_kwargs={'post_id': '<id>'},
        many=True, include_resource_linkage=True,
        type_='comments'
    )

    class Meta:
        type_ = 'posts'
        self_view = 'post_detail'
        self_view_kwargs = {'post_detail': '<id>'}
        self_view_many = 'posts_list'
```

See here for a full example.

CHAPTER 3

API Reference

3.1 API Reference

3.1.1 Core

class marshmallow_jsonapi.**Schema**(**args*, ***kwargs*)

Schema class that formats data according to JSON API 1.0. Must define the `type_` `class Meta` option.

Example:

```
from marshmallow_jsonapi import Schema, fields

def dasherize(text):
    return text.replace('_', '-')

class PostSchema(Schema):
    id = fields.Str(dump_only=True)  # Required
    title = fields.Str()

    author = fields.HyperlinkRelated(
        '/authors/{author_id}',
        url_kwargs={'author_id': '<author.id>'},
    )

    comments = fields.HyperlinkRelated(
        '/posts/{post_id}/comments',
        url_kwargs={'post_id': '<id>'},
        # Include resource linkage
        many=True, include_resource_linkage=True,
        type_='comments'
    )

    class Meta:
```

(continues on next page)

(continued from previous page)

```
        type_ = 'posts'  # Required
        inflect = dasherize
```

class Meta

Options object for *Schema*. Takes the same options as `marshmallow.Schema.Meta` with the addition of:

- `type_` - required, the JSON API resource type as a string.
- `inflect` - optional, an inflection function to modify attribute names.
- `self_url` - optional, URL to use to `self` in links
- `self_url_kwargs` - optional, replacement fields for `self_url`. String arguments enclosed in < > will be interpreted as attributes to pull from the schema data.
- `self_url_many` - optional, URL to use to `self` in top-level `links` when a collection of resources is returned.

check_relations(*relations*)

Recursive function which checks if a relation is valid.

format_error(*field_name*, *message*, *index=None*)

Override-able hook to format a single error message as an Error object.

See: http://jsonapi.org/format/#error-objects

format_errors(*errors*, *many*)

Format validation errors as JSON Error objects.

format_item(*item*)

Format a single datum as a Resource object.

See: http://jsonapi.org/format/#document-resource-objects

format_items(*data*, *many*)

Format data as a Resource object or list of Resource objects.

See: http://jsonapi.org/format/#document-resource-objects

format_json_api_response(*data*, *many*)

Post-dump hook that formats serialized data as a top-level JSON API object.

See: http://jsonapi.org/format/#document-top-level

generate_url(*link*, ***kwargs*)

Generate URL with any kwargs interpolated.

get_resource_links(*item*)

Hook for adding links to a resource object.

get_top_level_links(*data*, *many*)

Hook for adding links to the root of the response data.

inflect(*text*)

Inflect `text` if the `inflect` class Meta option is defined, otherwise do nothing.

on_bind_field(*field_name*, *field_obj*)

Schema hook override. When binding fields, set `data_key` (on marshmallow 3) or load_from (on marshmallow 2) to the inflected form of field_name.

wrap_response(*data*, *many*)

Wrap data and links according to the JSON API

3.1.2 Fields

Includes all the fields classes from `marshmallow.fields` as well as fields for serializing JSON API-formatted hyperlinks.

class marshmallow_jsonapi.fields.**BaseRelationship**(*default=<marshmallow.missing>*, *attribute=None*, *data_key=None*, *error=None*, *validate=None*, *required=False*, *allow_none=None*, *load_only=False*, *dump_only=False*, *missing=<marshmallow.missing>*, *error_messages=None*, ***metadata*)

Base relationship field.

This is used by *marshmallow_jsonapi.Schema* to determine which fields should be formatted as relationship objects.

See: http://jsonapi.org/format/#document-resource-object-relationships

class marshmallow_jsonapi.fields.**DocumentMeta**(***kwargs*)

Field which serializes to a "meta object" within a document's "top level".

Examples:

```
from marshmallow_jsonapi import Schema, fields

class UserSchema(Schema):
    id = fields.String()
    metadata = fields.DocumentMeta()

    class Meta:
        type_ = 'product'
        strict = True
```

See: http://jsonapi.org/format/#document-meta

class marshmallow_jsonapi.fields.**Meta**(***kwargs*)

Field which serializes to a "meta object" within a document's "top level".

Deprecated since version 0.18.0: Use *DocumentMeta* instead.

Examples:

```
from marshmallow_jsonapi import Schema, fields

class UserSchema(Schema):
    id = fields.String()
    metadata = fields.Meta()

    class Meta:
        type_ = 'product'
        strict = True
```

See: http://jsonapi.org/format/#document-meta

class marshmallow_jsonapi.fields.**Relationship**(*related_url=''*, *related_url_kwargs=None*, *self_url=''*, *self_url_kwargs=None*, *include_resource_linkage=False*, *schema=None*, *many=False*, *type_=None*, *id_field=None*, ***kwargs*)

Framework-independent field which serializes to a "relationship object".

See: http://jsonapi.org/format/#document-resource-object-relationships

Examples:

```
author = Relationship(
    related_url='/authors/{author_id}',
    related_url_kwargs={'author_id': '<author.id>'},
)

comments = Relationship(
    related_url='/posts/{post_id}/comments/',
    related_url_kwargs={'post_id': '<id>'},
    many=True, include_resource_linkage=True,
    type_='comments'
)
```

This field is read-only by default.

Parameters

- **related_url** (*str*) – Format string for related resource links.
- **related_url_kwargs** (*dict*) – Replacement fields for `related_url`. String arguments enclosed in `< >` will be interpreted as attributes to pull from the target object.
- **self_url** (*str*) – Format string for self relationship links.
- **self_url_kwargs** (*dict*) – Replacement fields for `self_url`. String arguments enclosed in `< >` will be interpreted as attributes to pull from the target object.
- **include_resource_linkage** (*bool*) – Whether to include a resource linkage (http://jsonapi.org/format/#document-resource-object-linkage) in the serialized result.
- **schema** (`marshmallow_jsonapi.Schema`) – The schema to render the included data with.
- **many** (*bool*) – Whether the relationship represents a many-to-one or many-to-many relationship. Only affects serialization of the resource linkage.
- **type** (*str*) – The type of resource.
- **id_field** (*str*) – Attribute name to pull ids from if a resource linkage is included.

deserialize(*value*, *attr=None*, *data=None*)

Deserialize `value`.

Raises **ValidationError** – If the value is not type `dict`, if the value does not contain a `data` key, and if the value is required but unspecified.

extract_value(*data*)

Extract the id key and validate the request structure.

class marshmallow_jsonapi.fields.**ResourceMeta**(***kwargs*)

Field which serializes to a "meta object" within a "resource object".

Examples:

```
from marshmallow_jsonapi import Schema, fields

class UserSchema(Schema):
    id = fields.String()
    meta_resource = fields.ResourceMeta()

    class Meta:
        type_ = 'product'
        strict = True
```

See: http://jsonapi.org/format/#document-resource-objects

3.1.3 Flask

Flask integration that avoids the need to hard-code URLs for links.

This includes a Flask-specific schema with custom Meta options and a relationship field for linking to related resources.

class marshmallow_jsonapi.flask.**Relationship**(*related_view=None*, *related_view_kwargs=None*, *self_view=None*, *self_view_kwargs=None*, ***kwargs*)

Field which serializes to a "relationship object" with a "related resource link".

See: http://jsonapi.org/format/#document-resource-object-relationships

Examples:

```
author = Relationship(
    related_view='author_detail',
    related_view_kwargs={'author_id': '<author.id>'},
)

comments = Relationship(
    related_view='posts_comments',
    related_view_kwargs={'post_id': '<id>'},
    many=True, include_resource_linkage=True,
    type_='comments'
)
```

This field is read-only by default.

Parameters

- **related_view** (*str*) – View name for related resource link.
- **related_view_kwargs** (*dict*) – Path kwargs fields for `related_view`. String arguments enclosed in < > will be interpreted as attributes to pull from the target object.
- **self_view** (*str*) – View name for self relationship link.
- **self_view_kwargs** (*dict*) – Path kwargs for `self_view`. String arguments enclosed in < > will be interpreted as attributes to pull from the target object.
- ****kwargs** – Same keyword arguments as *marshmallow_jsonapi.fields.Relationship*.

class marshmallow_jsonapi.flask.**Schema**(**args*, ***kwargs*)

A Flask specific schema that resolves self URLs from view names.

class Meta
Options object that takes the same options as `marshmallow-jsonapi.Schema`, but instead of `self_url`, `self_url_kwargs` and `self_url_many` has the following options to resolve the URLs from Flask views:

- `self_view` - View name to resolve the self URL link from.
- `self_view_kwargs` - Replacement fields for `self_view`. String attributes enclosed in `< >` will be interpreted as attributes to pull from the schema data.
- `self_view_many` - View name to resolve the self URL link when a collection of resources is returned.

OPTIONS_CLASS
alias of *SchemaOpts*

generate_url(*view_name*, ***kwargs*)
Generate URL with any kwargs interpolated.

class marshmallow_jsonapi.flask.**SchemaOpts**(*meta*, **args*, ***kwargs*)
Options to use Flask view names instead of hard coding URLs.

3.1.4 Exceptions

Exception classes.

exception marshmallow_jsonapi.exceptions.**IncorrectTypeError**(*message=None*, *actual=None*, *expected=None*)
Raised when client provides an invalid `type` in a request.

messages
JSON API-formatted error representation.

exception marshmallow_jsonapi.exceptions.**JSONAPIError**
Base class for all exceptions in this package.

3.1.5 Utilities

Utility functions.

This module should be considered private API.

marshmallow_jsonapi.utils.**resolve_params**(*obj*, *params*, *default=<marshmallow.missing>*)
Given a dictionary of keyword arguments, return the same dictionary except with values enclosed in `< >` resolved to attributes on `obj`.

marshmallow_jsonapi.utils.**tpl**(*val*)
Return value within `< >` if possible, else return `None`.

CHAPTER 4

Project info

4.1 Changelog

4.1.1 0.20.0 (2018-06-10)

Bug fixes:

- Fix serialization of `id` for `Relationship` fields when `attribute` is set (#69). Thanks @jordal for reporting and thanks @scottwernervt for the fix.

Note: The above fix could break some code that set `Relationship.id_field` before instantiating it. Set `Relationship.default_id_field` instead.

```
# before
fields.Relationship.id_field = 'item_id'

# after
fields.Relationship.default_id_field = 'item_id'
```

Support:

- Test refactoring and various doc improvements (#63, #86, #121# and #122). Thanks @scottwernervt.

4.1.2 0.19.0 (2018-05-27)

Features:

- Schemas passed to `fields.Relationship` will inherit context from the parent schema (#84). Thanks @asteinlein and @scottwernervt for the PRs.

4.1.3 0.18.0 (2018-05-19)

Features:

- Add `fields.ResourceMeta` for serializing a resource-level meta object (:issue:‘‘). Thanks @scottwernervt.

Other changes:

- *Backwards-incompatible*: Drop official support for Python 3.4.

4.1.4 0.17.0 (2018-04-29)

Features:

- Add support for marshmallow 3 (#97). Thanks @rockmnew.
- Thanks @mdodsworth for helping with #101.
- Move meta information object to document top level (#95). Thanks @scottwernervt.

4.1.5 0.16.0 (2017-11-08)

Features:

- Add support for exluding or including nested fields on relationships (#94). Thanks @scottwernervt for the PR.

Other changes:

- *Backwards-incompatible*: Drop support for marshmallow<2.8.0

4.1.6 0.15.1 (2017-08-23)

Bug fixes:

- Fix pointer for `id` in error objects (#90). Thanks @rgant for the catch and patch.

4.1.7 0.15.0 (2017-06-27)

Features:

- `Relationship` field supports deserializing included data (#83). Thanks @anuragagarwal561994 for the suggestion and thanks @asteinlein for the PR.

4.1.8 0.14.0 (2017-04-30)

Features:

- `Relationship` respects its passed `Schema's get_attribute` method when getting the `id` field for resource linkages (#80). Thanks @scmmmh for the PR.

4.1.9 0.13.0 (2017-04-18)

Features:

- Add support for including deeply nested relationships in compount documents (#61). Thanks @mrhanky17 for the PR.

4.1.10 0.12.0 (2017-04-16)

Features:

- Use default attribute value instead of raising exception if relationship is `None` on `Relationship` field (#75). Thanks @akira-dev.

4.1.11 0.11.1 (2017-04-06)

Bug fixes:

- Fix formatting JSON pointer when serializing an invalid object at index 0 (#77). Thanks @danpoland for the catch and patch.

4.1.12 0.11.0 (2017-03-12)

Bug fixes:

- Fix compatibility with marshmallow 3.x.

Other changes:

- *Backwards-incompatible*: Remove unused `utils.get_value_or_raise` function.

4.1.13 0.10.2 (2017-03-08)

Bug fixes:

- Fix format of error object returned when `data` key is not included in input (#66). Thanks @RazerM.
- Fix serializing compound documents when `Relationship` is passed a schema class and `many=True` (#67). Thanks @danpoland for the catch and patch.

4.1.14 0.10.1 (2017-02-05)

Bug fixes:

- Serialize `None` and empty lists (`[]`) to valid JSON-API objects (#58). Thanks @rgant for reporting and sending a PR.

4.1.15 0.10.0 (2017-01-05)

Features:

- Add `fields.Meta` for (de)serializing `meta` data on resource objects (#28). Thanks @rubdos for the suggestion and initial work. Thanks @RazerM for the PR.

Other changes:

- Test against Python 3.6.

4.1.16 0.9.0 (2016-10-08)

Features:

- Add Flask-specific schema with class Meta options for self link generation: `self_view`, `self_view_kwargs`, and `self_view_many` (#51). Thanks @asteinlein.

Bug fixes:

- Fix formatting of validation error messages on newer versions of marshmallow.

Other changes:

- Drop official support for Python 3.3.

4.1.17 0.8.0 (2016-06-20)

Features:

- Add support for compound documents (#11). Thanks @Tim-Erwin and @woodb for implementing this.
- *Backwards-incompatible*: Remove `include_data` parameter from `Relationship`. Use `include_resource_linkage` instead.

4.1.18 0.7.1 (2016-05-08)

Bug fixes:

- Format correction for error objects (#47). Thanks @ZeeD26 for the PR.

4.1.19 0.7.0 (2016-04-03)

Features:

- Correctly format `messages` attribute of `ValidationError` raised when `type` key is missing in input (#43). Thanks @ZeeD26 for the catch and patch.
- JSON pointers for error objects for relationships will point to the `data` key (#41). Thanks @cmanallen for the PR.

4.1.20 0.6.0 (2016-03-24)

Features:

- `Relationship` deserialization improvements: properly validate to-one and to-many relatinoships and validate the presense of the `data` key (#37). Thanks @cmanallen for the PR.
- `attributes` is no longer a required key in the `data` object (##39, #42). Thanks @ZeeD26 for reporting and @cmanallen for the PR.
- Added `id` serialization (#39). Thanks again @cmanallen.

4.1.21 0.5.0 (2016-02-08)

Features:

- Add relationship deserialization (#15).
- Allow serialization of foreign key attributes (#32).
- Relationship IDs serialize to strings, as is required by JSON-API (#31).
- `Relationship` field respects `dump_to` parameter (#33).

Thanks @cmanallen for all of these changes.

Other changes:

- The minimum supported marshmallow version is 2.3.0.

4.1.22 0.4.2 (2015-12-21)

Bug fixes:

- Relationship names are inflected when appropriate (#22). Thanks @angelosarto for reporting.

4.1.23 0.4.1 (2015-12-19)

Bug fixes:

- Fix serializing null and empty relationships with `flask.Relationship` (#24). Thanks @floqqi for the catch and patch.

4.1.24 0.4.0 (2015-12-06)

- Correctly serialize null and empty relationships (#10). Thanks @jo-tham for the PR.
- Add `self_url`, `self_url_kwargs`, and `self_url_many` class Meta options for adding `self` links. Thanks @asteinlein for the PR.

4.1.25 0.3.0 (2015-10-18)

- *Backwards-incompatible*: Replace `HyperlinkRelated` with `Relationship` field. Supports related links (`related`), relationship links (`self`), and resource linkages.
- *Backwards-incompatible*: Validate and deserialize JSON API-formatted request payloads.
- Fix error formatting when `many=True`.
- Fix error formatting in strict mode.

4.1.26 0.2.2 (2015-09-26)

- Fix for marshmallow 2.0.0 compat.

4.1.27 0.2.1 (2015-09-16)

- Compatibility with marshmallow>=2.0.0rc2.

4.1.28 0.2.0 (2015-09-13)

Features:

- Add framework-independent `HyperlinkRelated` field.
- Support inflection of attribute names via the `inflect` class Meta option.

Bug fixes:

- Fix for making `HyperlinkRelated` read-only by defualt.

Support:

- Docs updates.
- Tested on Python 3.5.

4.1.29 0.1.0 (2015-09-12)

- First PyPI release.
- Include Schema that serializes objects to resource objects.
- Flask-compatible HyperlinkRelate field for serializing relationships.
- Errors are formatted as JSON API errror objects.

4.2 Authors

4.2.1 Lead

- Steven Loria @sloria

4.2.2 Contributors (chronological)

- Jotham Apaloo @jo-tham
- Anders Steinlein @asteinlein
- @floqqi
- Colton Allen @cmanallen
- Dominik Steinberger @ZeeD26
- Tim Mundt @Tim-Erwin
- Brandon Wood @woodb
- Frazer McLean @RazerM
- J Rob Gant @rgant
- Dan Poland @danpoland

- Pierre CHAISY @akira-dev
- @mrhanky17
- Mark Hall @scmmmh
- Scott Werner @scottwernervt
- Michael Dodsworth @mdodsworth

4.3 Contributing Guidelines

4.3.1 In General

- PEP 8, when sensible.
- Test ruthlessly. Write docs for new features.
- Even more important than Test-Driven Development–*Human-Driven Development*.

4.3.2 In Particular

Questions, Feature Requests, Bug Reports, and Feedback…

…should all be reported on the Github Issue Tracker .

Setting Up for Local Development

1. Fork marshmallow-jsonapi on Github.

```
$ git clone https://github.com/marshmallow-code/marshmallow-jsonapi.git
$ cd marshmallow-jsonapi
```

2. Install development requirements. It is highly recommended that you use a virtualenv.

```
# After activating your virtualenv
$ pip install -r dev-requirements.txt
```

3. Install marshmallow-jsonapi in develop mode.

```
$ pip install -e .
```

Git Branch Structure

Marshmallow abides by the following branching model:

`dev` Current development branch. **New features should branch off here**.

`X.Y-line` Maintenance branch for release `X.Y`. **Bug fixes should be sent to the most recent release branch.** The maintainer will forward-port the fix to `dev`. Note: exceptions may be made for bug fixes that introduce large code changes.

Always make a new branch for your work, no matter how small. Also, **do not put unrelated changes in the same branch or pull request**. This makes it more difficult to merge your changes.

Pull Requests

1. Create a new local branch.

```
$ git checkout -b name-of-feature dev
```

2. Commit your changes. Write good commit messages.

```
$ git commit -m "Detailed commit message"
$ git push origin name-of-feature
```

3. Before submitting a pull request, check the following:

- If the pull request adds functionality, it is tested and the docs are updated.
- You've added yourself to `AUTHORS.rst`.

4. Submit a pull request to `marshmallow-code:dev` or the appropriate maintenance branch. The Travis CI build must be passing before your pull request is merged.

Running tests

To run all tests:

```
$ invoke test
```

To run tests on Python 2.7, 3.5, and 3.6 virtual environments (must have each interpreter installed):

```
$ tox
```

Documentation

Contributions to the documentation are welcome. Documentation is written in reStructured Text (rST). A quick rST reference can be found here. Builds are powered by Sphinx.

To install the packages for building the docs:

```
$ pip install -r docs/requirements.txt
```

To build the docs:

```
$ invoke docs -b
```

The `-b` (for "browse") automatically opens up the docs in your browser after building.

4.4 License

```
Copyright 2015-2017 Steven Loria

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
```

(continues on next page)

(continued from previous page)

```
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

CHAPTER 5

Links

- marshmallow-jsonapi @ GitHub
- marshmallow-jsonapi @ PyPI
- Issue Tracker

Python Module Index

m

marshmallow_jsonapi, 15
marshmallow_jsonapi.exceptions, 20
marshmallow_jsonapi.fields, 17
marshmallow_jsonapi.flask, 19
marshmallow_jsonapi.utils, 20

Index

B

BaseRelationship (class in marshmallow_jsonapi.fields), 17

C

check_relations() (marshmallow_jsonapi.Schema method), 16

D

deserialize() (marshmallow_jsonapi.fields.Relationship method), 18
DocumentMeta (class in marshmallow_jsonapi.fields), 17

E

extract_value() (marshmallow_jsonapi.fields.Relationship method), 18

F

format_error() (marshmallow_jsonapi.Schema method), 16
format_errors() (marshmallow_jsonapi.Schema method), 16
format_item() (marshmallow_jsonapi.Schema method), 16
format_items() (marshmallow_jsonapi.Schema method), 16
format_json_api_response() (marshmallow_jsonapi.Schema method), 16

G

generate_url() (marshmallow_jsonapi.flask.Schema method), 20
generate_url() (marshmallow_jsonapi.Schema method), 16
get_resource_links() (marshmallow_jsonapi.Schema method), 16
get_top_level_links() (marshmallow_jsonapi.Schema method), 16

I

IncorrectTypeError, 20
inflect() (marshmallow_jsonapi.Schema method), 16

J

JSONAPIError, 20

M

marshmallow_jsonapi (module), 15
marshmallow_jsonapi.exceptions (module), 20
marshmallow_jsonapi.fields (module), 17
marshmallow_jsonapi.flask (module), 19
marshmallow_jsonapi.utils (module), 20
messages (marshmallow_jsonapi.exceptions.IncorrectTypeError attribute), 20
Meta (class in marshmallow_jsonapi.fields), 17

O

on_bind_field() (marshmallow_jsonapi.Schema method), 16
OPTIONS_CLASS (marshmallow_jsonapi.flask.Schema attribute), 20

R

Relationship (class in marshmallow_jsonapi.fields), 17
Relationship (class in marshmallow_jsonapi.flask), 19
resolve_params() (in module marshmallow_jsonapi.utils), 20
ResourceMeta (class in marshmallow_jsonapi.fields), 18

S

Schema (class in marshmallow_jsonapi), 15
Schema (class in marshmallow_jsonapi.flask), 19
Schema.Meta (class in marshmallow_jsonapi), 16
Schema.Meta (class in marshmallow_jsonapi.flask), 19
SchemaOpts (class in marshmallow_jsonapi.flask), 20

T

tpl() (in module marshmallow_jsonapi.utils), 20

W

wrap_response() (marshmallow_jsonapi.Schema method), 16