calamus

Release 0.1.1

Swiss Data Science Center <contact@datascience.ch>

Aug 07, 2020

CONTENTS:

calamus is a library built on top of marshmallow to allow (de-)Serialization of Python classes to JSON-LD

CHAPTER

ONE

INSTALLATION

calamus releases and development versions are available from PyPI. You can install it using any tool that knows how to handle PyPI packages.

With pip:

```
$ pip install calamus
```

CHAPTER

TWO

USAGE

Assuming you have a class like

```
class Book:
    def __init__(self, _id, name):
        self._id = _id
        self.name = name
```

2.1 Declare schemes

You can declare a schema for serialization like

```
schema = fields.Namespace("http://schema.org/")

class BookSchema(JsonLDSchema):
    _id = fields.Id()
    name = fields.String(schema.name)

    class Meta:
        rdf_type = schema.Book
        model = Book
```

The `fields.Namespace` class represents an ontology namespace.

Make sure to set `rdf_type` to the RDF triple type you want get and `model` to the python class this schema applies to.

2.2 Serializing objects (“Dumping”)

You can now easily serialize python classes to JSON-LD

```
book = Book(_id="http://example.com/books/1", name="Ilias")
jsonld_dict = BookSchema().dump(book)
#{
#    "@id": "http://example.com/books/1",
#    "@type": "http://schema.org/Book",
#    "http://schema.org/name": "Ilias",
#}

jsonld_string = BookSchema().dumps(book)
#'{"@id": "http://example.com/books/1", "http://schema.org/name": "Ilias", "@type":
↪"http://schema.org/Book"}')
```

(continues on next page)

(continued from previous page)

2.3 Deserializing objects (“Loading”)

You can also easily deserialize JSON-LD to python objects

```
data = {
    "@id": "http://example.com/books/1",
    "@type": "http://schema.org/Book",
    "http://schema.org/name": "Ilias",
}
book = BookSchema().load(data)
#<Book(_id="http://example.com/books/1", name="Ilias")>
```

CHAPTER

THREE

SUPPORT

You can reach us on our Gitter Channel.

3.1 calamus API

3.1.1 Schema

Marshmallow schema implementation that supports JSON-LD.

class calamus.schema.**JsonLDSchema**(**args*, *only=None*, *exclude=()*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*, *unknown=None*, *flattened=False*, *lazy=False*, *_all_objects=None*, *_visited=None*, *_top_level=True*)

Schema for a JsonLD class.

Parameters

- **flattened** (*bool*) – If the JSON-LD should be loaded/dumped in flattened form
- **lazy** (*bool*) – Enables lazy loading of nested attributes

Example:

```
from calamus import JsonLDSchema
import calamus.fields as fields
from mymodels import User
schema = fields.Namespace("http://schema.org/")
class UserSchema(JsonLDSchema):
    class Meta:
        rdf_type = schema.Person
        model = User
    _id = fields.Id()
    birth_date = fields.Date(schema.birthDate)
    name = fields.String(schema.name)
```

OPTIONS_CLASS
alias of *JsonLDSchemaOpts*

get_reverse_links(*data: Mapping[str, Any]*, *field_name: str*)
Get all objects pointing to the object in data with the field field_name.

Used for unflattening a list.

make_instance(*data*, ***kwargs*)
Transform loaded dict into corresponding object.

class calamus.schema.**JsonLDSchemaMeta**(*name*, *bases*, *attrs*)
Meta-class for a for a JsonLDSchema class.

class calamus.schema.**JsonLDSchemaOpts**(*meta*, **args*, ***kwargs*)
Options class for *JsonLDSchema*.

Adds the following options:

- `rdf_type`: The RDF type(s) for this schema.
- `model`: The python type this schema (de-)serializes.
- `add_value_types`: Whether to add `@type` information to scalar field values.

3.1.2 Fields

Marshmallow fields for use with JSON-LD.

class calamus.fields.**BlankNodeId**(*name=None*)
A blank/anonymous node identifier.

Parameters **name** (*str*) – The name used to construct the blank node id. Default: `None`. Will use a UUID if not supplied.

class calamus.fields.**Boolean**(**args*, ***kwargs*)
A Boolean field.

class calamus.fields.**DateTime**(**args*, *extra_formats='%Y-%m-%d'*, ***kwargs*)
A date/time field.

class calamus.fields.**Dict**(*field_name=None*, **args*, ***kwargs*)
A dict field.

class calamus.fields.**Float**(**args*, ***kwargs*)
A float field.

class calamus.fields.**IRI**(**args*, ***kwargs*)
An external IRI reference.

class calamus.fields.**IRIReference**(*namespace*, *name*)
Represent an IRI in a namespace.

Parameters

- **namespace** (`Namespace`) – The `Namespace` this IRI is part of.
- **name** (*str*) – the property name of this IRI.

class calamus.fields.**Id**(**args*, ***kwargs*)
A node identifier.

class calamus.fields.**Integer**(**args*, ***kwargs*)
An integer field.

class calamus.fields.**List**(**args*, ***kwargs*)
A potentially ordered list using the `@list` keyword.

Parameters **ordered** (*bool*) – Whether this is an ordered (via `@list` keyword) list.

Warning: The JSON-LD flattening algorithm does not combine `@list` entries when merging nodes. So if you use `ordered=True` and flatten the output, and you have the node containing the list in multiple places in the graph, the node will get merged but its lists wont get merged (you get a list of lists instead), which means that the output can't be deserialized back to python objects.

property opts
Return parent's opts.

class calamus.fields.**Namespace**(*namespace*)
Represents a namespace/ontology.

Parameters **namespace** (*str*) – The base namespace URI for this namespace.

class calamus.fields.**Nested**(**args*, ***kwargs*)
A reference to one or more nested classes.

load_single_entry(*value*, *partial*)
Loads a single nested entry from its schema.

property schema
The nested Schema object.

This method was copied from marshmallow and modified to support multiple different nested schemes.

class calamus.fields.**Raw**(*field_name=None*, **args*, ***kwargs*)
A raw field.

class calamus.fields.**String**(**args*, ***kwargs*)
A string field.

3.1.3 Utilities

Calamus utilities.

calamus.utils.**normalize_id**(*id_object: Union[Mapping[str, Any], Iterable[Mapping[str, Any]], str]*)
Turns a JsonLD id reference into normalized form (list of strings).

calamus.utils.**normalize_type**(*type_data*)
Normalizes a JsonLD type reference as list of string.

calamus.utils.**normalize_value**(*value*)
Normalizes a JsonLD value object to a simple value.

3.2 License

3.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.3.1 Types of Contributions

Report issues / contacting developers

Report bugs on our issue tracker.

If you want to submit a bug, improvement or feature suggestions feel free to open a corresponding issue on GitHub.

If you are reporting a bug, please help us to speed up the diagnosing a problem by providing us with as much as information as possible. Ideally, that would include a step by step process on how to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for proposal discussions or epics and feel free to express your proposal on the topic. Once topic has been flushed out and we have decided how feature should be implemented, we can start implementing them.

Improvement requests

If you see room for improvement, please open an issue with a suggestion. Please motivate your suggestion by illustrating a problem it solves.

Write Documentation

calamus could always use more documentation, whether as part of the official calamus docs, in doc strings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/SwissDataScienceCenter/calamus/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.3.2 Get Started!

Ready to contribute? Here's how to set up *calamus* for local development.

1. Fork the *SwissDataScienceCenter/calamus* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/calamus.git
```

3. Ensure you have your development environment set up. For this we encourage usage of *poetry*:

```
$ poetry install
$ poetry shell
```

4. Create a branch for local development:

```
$ git checkout -b <issue_number>_<short_description>
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ poetry run pytest
```

   Before you submit a pull request, please reformat the code using black.

```
$ black
```

   You may want to set up black styling as a pre-commit hook to do this automatically:

```
$ poetry run pre-commit install
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s
    -m "type(scope): title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.3.3 Commit message guidelines

This project is using conventional commits style for generation of changelog upon each release. Therefore, it's important that our commit messages convey what they do correctly. Commit message should always follow this pattern:

    $ %{type}: %{description}

Type's used for describing commit's which will end up in changelog are `fix:` & `feat:`.

Please note that the `fix` type here is only for user-facing bug fixes and not fixes on tests or CI. For those, please use: `ci:` or `test:`

Full list of types which are in use:

- `feat:` - Used for new user-facing features.
- `fix:` - Used for fixing user-facing bugs.
- `chore:` - Used for changes which are not user-facing.
- `tests:` - Used for fixing existing or adding new tests.
- `docs:` - Used for adding more documentation.
- `refactor` - Used for changing the code structure.

3.3.4 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

- Make sure you agree with the license and follow the legal matter.
- The pull request should include tests and must not decrease test coverage.
- If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a doc string.
- The pull request should work for Python 3.6, 3.7 and 3.8. Check GitHub action builds and make sure that the tests pass for all supported Python versions.

3.4 Changes

3.4.1 0.3.2 (2020-08-07)

Fixes

- Fixed an issue where deserializing models with keyword arguments in their constructor raises an exception. (PR #39)

3.4.2 0.3.1 (2020-08-03)

Features

- Added a `Dict` field that simply passes along the contained dictionary (which should be valid Json-LD already). Added a `Raw` field that just returns the contained value. Added support for `add_value_types` at the field level. (PR #31)

Fixes

- Fixed an issue with sorting of schemes inside `Nested` fields. Original sort order is now preserved to allow users to specify precedence (in case of ambiguous types). Fixed the propagation of the `flattened` keyword to child schemes. Fixed `List` field deserialization calling the wrong `super()` method. (PR #31)

3.4.3 0.3.0 (2020-06-30)

Features

- Added lazy loading support (#12)

3.4.4 0.2.0 (2020-05-08)

Features

- Added IRIField (#24)
- Added BooleanField (1a93bdd)
- Added `init_name` setting to fields for cases where the name of the property differs from the name in the `__init__` method

Fixes

- Fixed an issue where `fields.Nested` would not work when used inside `fields.List`

3.4.5 0.1.2 (2020-05-08)

Features

- Allow serializing to a flat list (#5) (4289d86)
- Allow deserializing from a flat list (#4) (e8d56b3)

3.4.6 0.1.1 (2020-05-01)

Features

- Initial public release of calamus.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX