serdelicacy

Release 0.18.1

Samuel Roeca

Dec 06, 2020

TABLE OF CONTENTS

Goal: load and dump data into strongly-typed data structures.

1. "Deserialize" unstructured Python types into structured, type-hinted Python types (dataclasses.dataclass, typing.NamedTuples).
2. "Serialize" structured, type-hinted Python objects into unstructured Python types (eg, the reverse).

The top-level module contains *serdelicacy*'s public API and is directly importable from *serdelicacy*. If you choose to import names from anything outside of this module, do so at your own risk. Code organization in the package submodules is subject to change at any time is not reflected in this library's semantic versioning strategy.

CHAPTER

ONE

DESERIALIZATION

This function is the entrypoint to *serdelicacy*'s deserialization process:

`serdelicacy.`**`load`**(*obj*, *constructor*, *typesafe_constructor=True*)

Deserialize an object into its constructor.

Parameters

- **`obj`** (*`Any`*) – the serialized object that we want to deserialize
- **`constructor`** (*`Type[T]`*) – the type into which we want serialize *obj*
- **`typesafe_constructor`** (*`bool`*) – special-case flag to ensure the provided top-level constructor is not one of several "unsafe" types. If this is *False*, no validation will be performed. Keep this *True* unless you have an excellent reason to override it.

Returns A recursively-filled, typesafe instance of *constructor*

Raises

- **`TypeError`** – if typesafe is True and a non-typesafe constructor is provided
- ***`serdelicacy.DeserializeError`*** – triggered by any deserialization error

Return type T

Example

This simple library example shows *serdelicacy.load*'s recursive capabilites, exposed through an extremely simple pure-Python interface.

```
from typing import List
from dataclasses import dataclass
import serdelicacy

DATA = {
    "name": "Great library",
    "year_founded": 2018,
    "books": [
        {
            "title": "Great Book",
            "author": "Smitty",
        },
        {
            "title": "Bad Book",
            "author": "Smotty",
        },
```

(continues on next page)

(continued from previous page)

```
    ],
}

@dataclass
class Book:
    title: str
    author: str

@dataclass:
class Library:
    name: str
    year_founded: int
    books: List[Book]

LIBRARY = serdelicacy.load(DATA, Book)

assert isinstance(LIBRARY, Library)
assert isinstance(LIBRARY.books[0], Book)
```

CHAPTER

TWO

SERIALIZATION

This function is the entrypoint to *serdelicacy*'s serialization process:

`serdelicacy.`**`dump`**(*obj*, *convert_missing_to_none=False*)

Serialize an object into a lesser-typed form.

Parameters

- **`obj`** (*`Any`*) – the object that you would like to serialize.
- **`convert_missing_to_none`** (*`bool`*) – flag indicating whether or not we should retain a missing property's key and convert its value to *None*. This will keep all keys when serializing. Useful if you want to keep all column names and assign value of *None* to missing columns.

Returns A serialized form of *obj*.

Raises **`sedelicacy.SerializeError`** – raised for any unhandled error

Return type Any

Notes

If serializing a dataclass and *transform_dump* metadata exists in a *dataclass*'s '*dataclasses.field*, its value is assumed to be function whose result is serialized before being passed recursively down the chain.

Serialize from an instance of *a* -> an instance of *b*:

dataclass -> *Dict*
NamedTuple -> *Dict*
Enum -> enum value
str -> *str*
Sequence -> *List*
Mapping -> *Dict*
MISSING -> *None* (if *convert_missing_to_none* is *True*)
MISSING filtered out (if *convert_missing_to_none* is *False*)
Anything else -> *itself*

CHAPTER

THREE

UTILITIES

You may use the following tools to:

1. Operate on *serdelicacy*-affecred Python values in a typesafe way
2. Augment *dataclasses.dataclass* to give *serdelicacy* more information about how to process data.

`serdelicacy.`**`get`**(*value*, *default*)

Return *value* unless it's *MISSING*, in which case return *default*.

Similar to *dict.get*, but operates on *OptionalProperty*, provides no default for *default*, and is typesafe.

Parameters

- **`value`** (*`Union[serdelicacy.typedefs.Missing, T]`*) – an *OptionalProperty*
- **`default`** (*`T`*) – a value of the same type as *value*'s inner non-*MISSING* type

Returns *default* if *value* is *MISSING*, otherwise *value*

Return type T

`serdelicacy.`**`is_missing`**(*value*)

Check whether *value* is *MISSING*

Prevents users from needing to import *MISSING* constant.

Parameters **`value`** (*`Any`*) – any Python object

Returns *True* if a value is *MISSING*, *False* otherwise

Return type bool

`class` `serdelicacy.`**`Override`**(*validate=<function _noreturn>*, *transform_load=<function _id>*, *transform_postload=<function _id>*, *transform_dump=<function _id>*)

User-defined overrides for serde with *datacalsses.dataclass*

Should be passed into the *metadata* argument to *dataclasses.field* via the "serdelicacy" key.

Parameters

- **`validate`** – a function that either returns *True* on positive validation / *False* on non-validation, or returns nothing at all and instead relies on the raising of exceptions to indicate whether validation passed for failed.
- **`transform_load`** – a function that, when deserializing, is evaluated on an object before the object is recursively examined.
- **`transform_postload`** – a function that, when deserializing, is evaluated on an object after the object has been recursively examined. When possible, the *transform_load* should

be preferred over *transform_postload*, but there are situations where *transform_postload* is useful.

- **`transform_dump`** – a function that, when serializing a dataclass, is called on its value before it is recursively serialized.

Example

The following example shows how a function, returning *True/False*, is passed to *serdelicacy.load* through the *metadata* parameter to *dataclasses.field*.

```
from dataclasses import dataclass, field
import serdelicacy
from serdelicacy import Override

BOOK_RAW = {"author": "sam"}

@dataclass
class Book:
    author: str = field(
        metadata={"serdelicacy": Override(validate=str.istitle)}
    )

BOOK = serdelicacy.load(BOOK_RAW, Book)
```

This example should raise a *serdelicacy.DeserializeError*

CHAPTER

FOUR

TYPES

The following types are custom to *serdelicacy*:

serdelicacy.**OptionalProperty = typing.Union[serdelicacy.typedefs.Missing, ~T]**

Type alias for a value that can be MISSING

OptionalProperty is extremely useful when differentiating between values that can be missing and those that are *None*.

Example

```
from typing import List, Optional
from dataclasses import dataclass
import serdelicacy
from serdelicacy import OptionalProperty

DATA = [
    {
        "hello": "friend",
        "world": "foe",
    },
    {
        "hello": "Rawr",
    },
    {
        "hello": None,
        "world": "hehe",
    },
]

@dataclass
class HelloWorld:
    hello: Optional[str]
    world: OptionalProperty[str]

PARSED = serdelicacy.load(DATA, List[HelloWorld])

assert serdelicacy.is_missing(PARSED[1].world)
assert not PARSED[1].world  # note: it's Falsey!
assert PARSED[2].hello is None
```

CHAPTER

FIVE

EXCEPTIONS

serdelicacy may raise the following Python exceptions during the serialization and/or the deserialization process.

exception serdelicacy.**SerdeError**

Bases: `Exception`

Base error for *serdelicacy*.

Associated with both *serdelicacy.dump* and *serdelicacy.load* through *serdelicacy.DeserializeError* and *serdelicacy.SerializeError*.

exception serdelicacy.**DeserializeError**(*type_expected*, *value_received*, *depth*, *key*, *message_prefix=''*, *message_postfix=''*, *message_override=''*)

Bases: `serdelicacy.errors.SerdeError`

Deserialization error associated with *serdelicacy.load*.

Deserializing arbitrarily-nested JSON often results in opaque deserialization errors. This Exception class provides a clear, consistent debugging message.

Parameters

- **`type_expected`** (*`Type`*) – the type *serdelicacy* expected a value to be.
- **`value_received`** (*`Any`*) – the actual object received.
- **`depth`** (*`List[serdelicacy.errors.DepthContainer]`*) – objects containing information about the current level of recursion.
- **`key`** (*`Any`*) – if the current value is associated with a key, provide the key's value. This can technically have any value.
- **`message_prefix`** (*`str`*) – message to prepend to the generated error message.
- **`message_postfix`** (*`str`*) – message to postpend to the generated error message.
- **`message_override`** (*`str`*) – if provided, replaces generated error message.

Note: This is part of the public interface insofar as you may need it to catch errors. You won't need it to instantiate or raise this exception it yourself.

exception serdelicacy.**SerializeError**

Bases: `serdelicacy.errors.SerdeError`

Serialization error associated with *serdelicacy.dump*.

CHAPTER SIX

SERDELICACY

Serialize (`serdelicacy.dump`) and deserialize (`serdelicacy.load`) from/to strongly-typed, native Python data structures.

Read the latest documentation here

6.1 Features

1. Effortless deserialization of unstructured Python types into structured, type-hinted Python types (`dataclasses.dataclass`, `typing.NamedTuple`)
2. Effortless serialization of structured, type-hinted Python objects into unstructured Python types (eg, the reverse)
3. Clear error messages when serde fails at runtime
4. No inherited, non-standard types. dataclasses, NamedTuples, and other standard Python types are bread and butter
5. Editor support: I like my autocompletion, so I jump through lots of hoops to make this library compatible with Jedi
6. Handle optional properties with a domain-specific `serdelicacy.OptionalProperty`
7. Enable customization through sophisticated validation, deserialization overrides, and serialization overrides for dataclasses.
8. Require no 3rd party dependencies; Python 3.8+

6.2 Installation

```
# With pip
pip install serdelicacy

# With poetry
poetry add serdelicacy
```

6.3 Usage

See examples folder.

6.4 Validation / transformation for dataclasses

Customization override options are available for validations and transformations on both deserialization and serialization. Custom overrides are available for `dataclasses` through the `metadata` argument to the `dataclasses.field` function:

```python
from dataclasses import dataclass, field

import serdelicacy
from serdelicacy import Override

def _is_long_enough(value) -> None:
    if len(value) < 4:
        raise ValueError(f"'{value}' is not enough characters")

VALUE = {"firstname": "richard", "lastname": "spencerson"}

@dataclass
class Person:
    firstname: str = field(
        metadata={
            "serdelicacy": Override(
                validate=_is_long_enough,
                transform_load=str.title,
            )
        }
    )
    lastname: str = field(
        metadata={
            "serdelicacy": Override(
                validate=_is_long_enough,
                transform_load=str.title,
                transform_dump=str.upper,
            )
        }
    )

print(serdelicacy.load(VALUE, Person))
```

As suggested by the Python dataclasses.field documentation, all `serdelicacy`-related field metadata is namespaced to 1 dictionary key: `serdelicacy`. Its value should be of type `serdelicacy.Override`, a dataclass itself whose fields are the following:

- *validate*: *Callable[[Any], NoReturn], Callable[[Any], bool]*: a function that either a) returns a boolean where False indicates failed validation or b) nothing, but raises Python exceptions on validation failure. Is executed as the final step of a value's load, after all transformations have been completed. By default, this is a function that does nothing.
- *transform_load*: *Callable[[Any], Any]*. This transformation is executed before any other loading takes place. By default, this is an [identity function](https://en.wikipedia.org/wiki/Identity_function)
- *transform_postload*: this should be *Callable[[T], T]]*, where *T* is the type of the field. This transformation is executed after all recursive loading takes place as the final step before the value is returned for upstream processing. By default, this is an [identity function](https://en.wikipedia.org/wiki/Identity_function)
- *transform_dump*: this should be *Callable[[T], Any]]*, where *T* is the type of the field. This function is executed before a value is recursively serialized. By default, this is an [identity function](https://en.wikipedia.org/wiki/Identity_function)

Finally, you may not need to use these tools initially, but if you have strict validation or transformation requirements on your project, you'll be extremely happy they're here

6.5 FAQ

6.5.1 My JSON keys contain whitespace, etc

Simple solution: use `typeing.TypeDict`'s backwards-compatibility syntax.

```
from pprint import pprint
from typing import List, TypedDict

import serdelicacy
from serdelicacy import OptionalProperty

DATA = [
    {
        "weird, key": 1,
        "normal": 2,
    },
    {
        "normal": 3,
    },
]

DataItem = TypedDict(
    "DataItem",
    {
        "weird, key": OptionalProperty[int],
        "normal": int,
    },
)

LOADED = serdelicacy.load(DATA, List[DataItem])

print("Loaded data:")
pprint(LOADED)
```

(continues on next page)

(continued from previous page)

```
print("Re-serialized data:")
pprint(serdelicacy.dump(LOADED))
```

This prints the following to the console.

```
Loaded data:
[{'normal': 2, 'weird, key': 1},
 {'normal': 3, 'weird, key': <Missing property>}]
Re-serialized data:
[{'normal': 2, 'weird, key': 1}, {'normal': 3}]
```

Try changing values in your JSON data; you'll get runtime errors if your data does not conform to the above schema. Additionally, `mypy` should call out any misused variable keys / types. In short, this has enabled a type-safe load and a perfectly sane dump.

6.6 Local Development

Local development for this project is simple.

Dependencies

Install the following tools manually.

- Poetry
- GNU Make

Recommended

- asdf

Set up development environment

```
make setup
```

Run Tests

```
make test
```

6.7 Notes

- Initially inspired by undictify and a PR I helped with. serdelicacy's goals are different; it's focused on serde instead of general function signature overrides.
- I also notice some striking similarities with a library called typedload (great minds think alike, I guess :p). I renamed my top-level functions to "load" and "dump" in typedload's homage. Unfortunately, as of version `1.20`, typedload does not handle all types of dataclasses elegantly (mainly, InitVar). Since typedload supports Python 3.5+, it never will elegantly handle all dataclasses without lots of unfortunate conditionals in the codebase. If you must use Python 3.7-, I suggest looking into typedload.

6.8 Written by

Samuel Roeca *samuel.roeca@gmail.com*

PYTHON MODULE INDEX

INDEX