py-serializable

Release 1.0.0-rc.1

Paul Horton

Jan 18, 2024

CONTENTS:

This Pythonic-library can be used to magically handle serialization of your Python Objects to JSON or XML and de-serialization from JSON or XML back to Pythonic Object instances.

This library relies upon your Python Classes utilising the @property decorator and can optionally take additional configuration which allows you to control how a class is (de-)serialized.

See also:

- Python's property() function/decorator

CHAPTER

ONE

GETTING STARTED

Let's work a simple example together.

I have a two Python classes that together I use to model Books. They are `Book` and `Chapter`, and they are defined as follows:

```
class Chapter:

    def __init__(self, *, number: int, title: str) -> None:
        self._number = number
        self._title = title

    @property
    def number(self) -> int:
        return self._number

    @property
    def title(self) -> str:
        return self._title

class Book:

    def __init__(self, *, title: str, isbn: str, edition: int, publish_date: date,␣
↪authors: Iterable[str],
                 chapters: Optional[Iterable[Chapter]] = None) -> None:
        self._title = title
        self._isbn = isbn
        self._edition = edition
        self._publish_date = publish_date
        self._authors = set(authors)
        self.chapters = chapters or []

    @property
    def title(self) -> str:
        return self._title

    @property
    def isbn(self) -> str:
        return self._isbn

    @property
    def edition(self) -> int:
```

(continues on next page)

(continued from previous page)

```
        return self._edition

    @property
    def publish_date(self) -> date:
        return self._publish_date

    @property
    def authors(self) -> Set[str]:
        return self._authors

    @property
    def chapters(self) -> List[Chapter]:
        return self._chapters

    @chapters.setter
    def chapters(self, chapters: Iterable[Chapter]) -> None:
        self._chapters = list(chapters)
```

To make a class serializable to/from JSON or XML, the class must be annotated with the decorator *serializable.serializable_class*.

By simply modifying the classes above, we make them (de-)serializable with this library (albeit with some default behaviour implied!).

This makes our classes:

```
import serializable

@serializable.serializable_class
class Chapter:

    def __init__(self, *, number: int, title: str) -> None:
        self._number = number
        self._title = title

    @property
    def number(self) -> int:
        return self._number

    @property
    def title(self) -> str:
        return self._title

@serializable.serializable_class
class Book:

    def __init__(self, *, title: str, isbn: str, edition: int, publish_date: date,␣
↪authors: Iterable[str],
                 chapters: Optional[Iterable[Chapter]] = None) -> None:
        self._title = title
        self._isbn = isbn
        self._edition = edition
        self._publish_date = publish_date
```

(continues on next page)

(continued from previous page)

```
        self._authors = set(authors)
        self.chapters = chapters or []

    @property
    def title(self) -> str:
        return self._title

    @property
    def isbn(self) -> str:
        return self._isbn

    @property
    def edition(self) -> int:
        return self._edition

    @property
    def publish_date(self) -> date:
        return self._publish_date

    @property
    def authors(self) -> Set[str]:
        return self._authors

    @property
    def chapters(self) -> List[Chapter]:
        return self._chapters

    @chapters.setter
    def chapters(self, chapters: Iterable[Chapter]) -> None:
        self._chapters = list(chapters)
```

At this point, we can serialize an instance of Book to JSON as follows:

```
book = Book(title="My Book", isbn="999-888777666555", edition=1, publish_date=datetime.
↪utcnow(), authors=['me'])
print(book.as_json())
```

which outputs:

```
{
    "title": "My Book",
    "isbn": "999-888777666555",
    "edition": 1,
    "publishDate": "2022-08-10",
    "authors": [
        "me"
    ]
}
```

We could also serialized to XML as follows:

```
print(book.as_xml())
```

which outputs:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
    <title>My Book</title>
    <isbn>999-888777666555</isbn>
    <edition>1</edition>
    <publishDate>2022-08-10</publishDate>
    <author>me</author>
</book>
```

CHAPTER

TWO

CUSTOMISING SERIALIZATION

There are various scenarios whereby you may want to have more control over the structure (particularly in XML) that is generated when serializing an object, and thus understanding how to deserialize JSON or XML back to an object.

This library provides a number of *meta methods* that you can override in your Python classes to achieve this.

2.1 Property Name Mappings

You can directly control mapping of property names for properties in a Class by adding the decorators *serializable.json_name()* or *serializable.xml_name()*.

For example, you might have a property called **isbn** in your class, but when serialized to JSON it should be called **isbn_number**.

To implement this mapping, you would alter your class as follows adding the *serializable.json_name()* decorator to the **isbn** property:

```
@serializable.serializable_class
class Book:

    def __init__(self, title: str, isbn: str, publish_date: date, authors: Iterable[str],
        ...

    @property
    @serializable.json_name('isbn_number')
    def isbn(self) -> str:
        return self._isbn
```

2.2 Excluding Property from Serialization

Properties can be ignored during deserialization by including them in the *serializable.serializable_class()* annotation as per the following example.

A typical use case for this might be where a JSON schema is referenced, but this is not part of the constructor for the class you are deserializing to.

```
@serializable.serializable_class(ignore_during_deserialization=['$schema'])
class Book:
  ...
```

2.3 Handling `None` Values

By default, `None` values will lead to a Property being excluded from the serialization process to keep the output as concise as possible. There are many cases (and schemas) where this is however not the required behaviour.

You can force a Property to be serialized even when the value is `None` by annotating as follows:

```
@serializable.include_none
def email(self) -> Optional[str]:
    return self._email
```

2.4 Customised Property Serialization

This feature allows you to handle, for example, serialization of `datetime.date` Python objects to and from strings.

Depending on your use case, the string format could vary, and thus this library makes no assumptions. We have provided an some example helpers for (de-)serializing dates and datetimes.

To define a custom serializer for a property, add the *serializable.type_mapping()* decorator to the property. For example, to have a property named *created* be use the *serializable.helpers.Iso8601Date* helper you would add the following method to your class:

```
@serializable.serializable_class
class Book:

    def __init__(self, title: str, isbn: str, publish_date: date, authors: Iterable[str],
        ...

    @property
    @serializable.type_mapping(Iso8601Date)
    def publish_date(self) -> date:
        return self._publish_date
```

2.4.1 Writing Custom Property Serializers

You can write your own custom property serializer. The only requirements are that it must extend *serializable.helpers.BaseHelper* and therefore implement the `serialize()` and `deserialize()` class methods.

For examples, see *serializable.helpers*.

2.5 Serializing Lists & Sets

Particularly in XML, there are many ways that properties which return Lists or Sets could be represented. We can handle this by adding the decorator *serializable.xml_array()* to the appropriate property in your class.

For example, given a Property that returns `Set[Chapter]`, this could be serialized in one of a number of ways:

Listing 1: Example 1: Nested list under a property name in JSON

```
{
    "chapters": [
        { /* chapter 1 here... */ },
        { /* chapter 2 here... */ },
        // etc...
    ]
}
```

Listing 2: Example 2: Nested list under a property name in XML

```
<chapters>
    <chapter><!-- chapter 1 here... --></chapter>
    <chapter><!-- chapter 2 here... --></chapter>
    <!-- etc... -->
</chapters>
```

Listing 3: Example 3: Collapsed list under a (potentially singular of the) property name in XML

```
<chapter><!-- chapter 1 here... --></chapter>
<chapter><!-- chapter 2 here... --></chapter>
```

As we have only identified one possible structure for JSON at this time, the implementation of only affects XML (de-)serialization at this time.

For *Example 2*, you would add the following to your class:

```
@property
@serializable.xml_array(XmlArraySerializationType.NESTED, 'chapter')
def chapters(self) -> List[Chapter]:
    return self._chapters
```

For *Example 3*, you would add the following to your class:

```
@property
@serializable.xml_array(XmlArraySerializationType.FLAT, 'chapter')
def chapters(self) -> List[Chapter]:
    return self._chapters
```

Further examples are available in our unit tests.

2.6 Serialization Views

Many object models can be serialized to and from multiple versions of a schema or different schemas. In `py-serialization` we refer to these as Views.

By default all Properties will be included in the serialization process, but this can be customised based on the View.

2.6.1 Defining Views

A View is a class that extends *serializable.ViewType* and you should create classes as required in your implementation.

For example:

```
from serializable import ViewType

class SchemaVersion1(ViewType):
   pass
```

2.6.2 Property Inclusion

Properties can be annotated with the Views for which they should be included.

For example:

```
@property
@serializable.view(SchemaVersion1)
def address(self) -> Optional[str]:
    return self._address
```

2.6.3 Handling `None` Values

Further to the above, you can vary the `None` value per View as follows:

```
@property
@serializable.include_none(SchemaVersion2)
@serializable.include_none(SchemaVersion3, "RUBBISH")
def email(self) -> Optional[str]:
    return self._email
```

The above example will result in `None` when serializing with the View `SchemaVersion2`, but the value `RUBBISH` when serializing to the View `SchemaVersion3` when `email` is not set.

2.6.4 Serializing For a View

To serialized for a specific View, include the View when you perform the serialisation.

Listing 4: JSON Example

```
ThePhoenixProject.as_json(view_=SchemaVersion1)
```

Listing 5: XML Example

```
ThePhoenixProject.as_xml(view_=SchemaVersion1)
```

2.7 XML Element Ordering

Some XML schemas utilise sequence which requires elements to be in a prescribed order.

You can control the order properties are serialized to elements in XML by utilising the *serializable.xml_sequence()* decorator. The default sort order applied to properties is 100 (where lower is earlier in the sequence).

In the example below, the `isbn` property will be output first.

```
@property
@serializable.xml_sequence(1)
def isbn(self) -> str:
    return self._isbn
```

CHAPTER

THREE

PROPERTY NAME FORMATTING

By default, `py-serializable` uses it's *serializable.formatters.CamelCasePropertyNameFormatter* formatter for translating actual Python property names to element names in either JSON or XML.

`py-serializable` includes a number of name formatters out of the box, but you can also create your own if required.

3.1 Included Formatters

`py-serializable` includes three common formatters out of the box.

1. Camel Case Formatter: *serializable.formatters.CamelCasePropertyNameFormatter* (the default)
2. Kebab Case Formatter: *serializable.formatters.KebabCasePropertyNameFormatter*
3. Snake Case Formatter: *serializable.formatters.SnakeCasePropertyNameFormatter*

A summary of how these differ is included in the below table.

Python Property Name	Camel Case	Kebab Case	Snake Case
books	books	books	books
big_book	bigBook	big-book	big_book
a_very_big_book	aVeryBigBook	a-very-big-book	a_very_big_book

3.2 Changing the Formatter

You can change the formatter being used by easily. The example below changes the formatter to be Snake Case.

```
from serializable.formatters import CurrentFormatter, SnakeCasePropertyNameFormatter

CurrentFormatter.formatter = SnakeCasePropertyNameFormatter
```

3.3 Custom Formatters

If none of the included formatters work for you, why not write your own?

The only requirement is that it extends *serializable.formatters.BaseNameFormatter*!

CHAPTER

FOUR

EXAMPLES

4.1 Models used in Unit Tests

```
# encoding: utf-8

# This file is part of py-serializable
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# SPDX-License-Identifier: Apache-2.0
# Copyright (c) Paul Horton. All Rights Reserved.

import re
from datetime import date
from decimal import Decimal
from enum import Enum, unique
from typing import Any, Dict, Iterable, List, Optional, Set, Type
from uuid import UUID, uuid4

import serializable
from serializable import ViewType, XmlArraySerializationType
from serializable.helpers import BaseHelper, Iso8601Date

"""
Model classes used in unit tests and examples.
"""


class SchemaVersion1(ViewType):
    pass
```

(continues on next page)

(continued from previous page)

```


class SchemaVersion2(ViewType):
    pass


class SchemaVersion3(ViewType):
    pass


class SchemaVersion4(ViewType):
    pass


SCHEMAVERSION_MAP: Dict[int, Type[ViewType]] = {
    1: SchemaVersion1,
    2: SchemaVersion2,
    3: SchemaVersion3,
    4: SchemaVersion4,
}


class ReferenceReferences(BaseHelper):

    @classmethod
    def serialize(cls, o: Any) -> Set[str]:
        if isinstance(o, set):
            return set(map(lambda i: str(i.ref), o))

        raise ValueError(f'Attempt to serialize a non-set: {o.__class__}')

    @classmethod
    def deserialize(cls, o: Any) -> Set['BookReference']:
        print(f'Deserializing {o} ({type(o)})')
        references: Set['BookReference'] = set()
        if isinstance(o, list):
            for v in o:
                references.add(BookReference(ref=v))
            return references

        raise ValueError(f'Attempt to deserialize a non-set: {o.__class__}')


class TitleMapper(BaseHelper):

    @classmethod
    def json_serialize(cls, o: str) -> str:
        return f'{{J}} {o}'

    @classmethod
    def json_deserialize(cls, o: str) -> str:
        return re.sub(r'^\{J} ', '', o)
```

(continues on next page)

(continued from previous page)

```
90
91      @classmethod
92      def xml_serialize(cls, o: str) -> str:
93          return f'{{X}} {o}'
94
95      @classmethod
96      def xml_deserialize(cls, o: str) -> str:
97          return re.sub(r'^\{X} ', '', o)
98
99
100 @serializable.serializable_class
101 class Chapter:
102
103     def __init__(self, *, number: int, title: str) -> None:
104         self._number = number
105         self._title = title
106
107     @property
108     def number(self) -> int:
109         return self._number
110
111     @property
112     def title(self) -> str:
113         return self._title
114
115     def __eq__(self, other: Any) -> bool:
116         if isinstance(other, Chapter):
117             return hash(other) == hash(self)
118         return False
119
120     def __hash__(self) -> int:
121         return hash((self.number, self.title))
122
123
124 @serializable.serializable_class
125 class Publisher:
126
127     def __init__(self, *, name: str, address: Optional[str] = None, email: Optional[str]␣
    ↪= None) -> None:
128         self._name = name
129         self._address = address
130         self._email = email
131
132     @property
133     def name(self) -> str:
134         return self._name
135
136     @property
137     @serializable.view(SchemaVersion2)
138     @serializable.view(SchemaVersion4)
139     def address(self) -> Optional[str]:
140         return self._address
```

(continues on next page)

(continued from previous page)

```
    @property
    @serializable.include_none(SchemaVersion2)
    @serializable.include_none(SchemaVersion3, 'RUBBISH')
    def email(self) -> Optional[str]:
        return self._email

    def __eq__(self, other: object) -> bool:
        if isinstance(other, Publisher):
            return hash(other) == hash(self)
        return False

    def __hash__(self) -> int:
        return hash((self.name, self.address, self.email))


@unique
class BookType(Enum):
    FICTION = 'fiction'
    NON_FICTION = 'non-fiction'


@serializable.serializable_class(name='edition')
class BookEdition:

    def __init__(self, *, number: int, name: str) -> None:
        self._number = number
        self._name = name

    @property
    @serializable.xml_attribute()
    def number(self) -> int:
        return self._number

    @property
    @serializable.xml_name('.')
    def name(self) -> str:
        return self._name

    def __eq__(self, other: object) -> bool:
        if isinstance(other, BookEdition):
            return hash(other) == hash(self)
        return False

    def __hash__(self) -> int:
        return hash((self.number, self.name))


@serializable.serializable_class
class BookReference:

    def __init__(self, *, ref: str, references: Optional[Iterable['BookReference']] =␣
```

(continues on next page)

(continued from previous page)

```
↪None) -> None:
193         self.ref = ref
194         self.references = set(references or {})
195 
196     @property
197     @serializable.json_name('reference')
198     @serializable.xml_attribute()
199     def ref(self) -> str:
200         return self._ref
201 
202     @ref.setter
203     def ref(self, ref: str) -> None:
204         self._ref = ref
205 
206     @property
207     @serializable.json_name('refersTo')
208     @serializable.type_mapping(ReferenceReferences)
209     @serializable.xml_array(serializable.XmlArraySerializationType.FLAT, 'reference')
210     def references(self) -> Set['BookReference']:
211         return self._references
212 
213     @references.setter
214     def references(self, references: Iterable['BookReference']) -> None:
215         self._references = set(references)
216 
217     def __eq__(self, other: object) -> bool:
218         if isinstance(other, BookReference):
219             return hash(other) == hash(self)
220         return False
221 
222     def __hash__(self) -> int:
223         return hash((self.ref, tuple(self.references)))
224 
225     def __repr__(self) -> str:
226         return f'<BookReference ref={self.ref}, targets={len(self.references)}>'
227 
228 
229 @serializable.serializable_class(name='bigbook',
230                                  ignore_during_deserialization=['something_to_be_ignored
↪', 'ignore_me', 'ignored'])
231 class Book:
232 
233     def __init__(self, title: str, isbn: str, publish_date: date, authors: Iterable[str],
234                  publisher: Optional[Publisher] = None, chapters:␣
↪Optional[Iterable[Chapter]] = None,
235                  edition: Optional[BookEdition] = None, type: BookType = BookType.
↪FICTION,
236                  id: Optional[UUID] = None, references:␣
↪Optional[Iterable[BookReference]] = None,
237                  rating: Optional[Decimal] = None) -> None:
238         self._id = id or uuid4()
239         self._title = title
```

(continues on next page)

(continued from previous page)

```
        self._isbn = isbn
        self._edition = edition
        self._publish_date = publish_date
        self._authors = set(authors)
        self._publisher = publisher
        self.chapters = list(chapters or [])
        self._type = type
        self.references = set(references or [])
        self.rating = Decimal('NaN') if rating is None else rating

    @property
    @serializable.xml_sequence(1)
    def id(self) -> UUID:
        return self._id

    @property
    @serializable.xml_sequence(2)
    @serializable.type_mapping(TitleMapper)
    def title(self) -> str:
        return self._title

    @property
    @serializable.json_name('isbn_number')
    @serializable.xml_attribute()
    @serializable.xml_name('isbn_number')
    def isbn(self) -> str:
        return self._isbn

    @property
    @serializable.xml_sequence(3)
    def edition(self) -> Optional[BookEdition]:
        return self._edition

    @property
    @serializable.xml_sequence(4)
    @serializable.type_mapping(Iso8601Date)
    def publish_date(self) -> date:
        return self._publish_date

    @property
    @serializable.xml_array(XmlArraySerializationType.FLAT, 'author')
    @serializable.xml_sequence(5)
    def authors(self) -> Set[str]:
        return self._authors

    @property
    @serializable.xml_sequence(7)
    def publisher(self) -> Optional[Publisher]:
        return self._publisher

    @property
    @serializable.xml_array(XmlArraySerializationType.NESTED, 'chapter')
```

(continues on next page)

(continued from previous page)

```
    @serializable.xml_sequence(8)
    def chapters(self) -> List[Chapter]:
        return self._chapters

    @chapters.setter
    def chapters(self, chapters: Iterable[Chapter]) -> None:
        self._chapters = list(chapters)

    @property
    @serializable.xml_sequence(6)
    def type(self) -> BookType:
        return self._type

    @property
    @serializable.view(SchemaVersion4)
    @serializable.xml_array(serializable.XmlArraySerializationType.NESTED, 'reference')
    @serializable.xml_sequence(7)
    def references(self) -> Set[BookReference]:
        return self._references

    @references.setter
    def references(self, references: Iterable[BookReference]) -> None:
        self._references = set(references)

    @property
    @serializable.xml_sequence(20)
    def rating(self) -> Decimal:
        return self._rating

    @rating.setter
    def rating(self, rating: Decimal) -> None:
        self._rating = rating


ThePhoenixProject_v1 = Book(
    title='The Phoenix Project', isbn='978-1942788294', publish_date=date(year=2018,␣
↪month=4, day=16),
    authors=['Gene Kim', 'Kevin Behr', 'George Spafford'],
    publisher=Publisher(name='IT Revolution Press LLC'),
    edition=BookEdition(number=5, name='5th Anniversary Limited Edition'),
    id=UUID('f3758bf0-0ff7-4366-a5e5-c209d4352b2d'),
    rating=Decimal('9.8')
)

ThePhoenixProject_v1.chapters.append(Chapter(number=1, title='Tuesday, September 2'))
ThePhoenixProject_v1.chapters.append(Chapter(number=2, title='Tuesday, September 2'))
ThePhoenixProject_v1.chapters.append(Chapter(number=3, title='Tuesday, September 2'))
ThePhoenixProject_v1.chapters.append(Chapter(number=4, title='Wednesday, September 3'))

ThePhoenixProject_v2 = Book(
    title='The Phoenix Project', isbn='978-1942788294', publish_date=date(year=2018,␣
↪month=4, day=16),
```

(continues on next page)

(continued from previous page)

```
    authors=['Gene Kim', 'Kevin Behr', 'George Spafford'],
    publisher=Publisher(name='IT Revolution Press LLC', address='10 Downing Street'),
    edition=BookEdition(number=5, name='5th Anniversary Limited Edition'),
    id=UUID('f3758bf0-0ff7-4366-a5e5-c209d4352b2d'),
    rating=Decimal('9.8')
)

ThePhoenixProject_v2.chapters.append(Chapter(number=1, title='Tuesday, September 2'))
ThePhoenixProject_v2.chapters.append(Chapter(number=2, title='Tuesday, September 2'))
ThePhoenixProject_v2.chapters.append(Chapter(number=3, title='Tuesday, September 2'))
ThePhoenixProject_v2.chapters.append(Chapter(number=4, title='Wednesday, September 3'))

SubRef1 = BookReference(ref='sub-ref-1')
SubRef2 = BookReference(ref='sub-ref-2')
SubRef3 = BookReference(ref='sub-ref-3')

Ref1 = BookReference(ref='my-ref-1')
Ref2 = BookReference(ref='my-ref-2', references=[SubRef1, SubRef3])
Ref3 = BookReference(ref='my-ref-3', references=[SubRef2])

ThePhoenixProject_v2.references = {Ref3, Ref2, Ref1}

ThePhoenixProject = ThePhoenixProject_v2

if __name__ == '__main__':
    tpp_as_xml = ThePhoenixProject.as_xml()  # type:ignore[attr-defined]
    tpp_as_json = ThePhoenixProject.as_json()  # type:ignore[attr-defined]
    print(tpp_as_xml, tpp_as_json, sep='\n\n')

    import io
    import json
    tpp_from_xml = ThePhoenixProject.from_xml(  # type:ignore[attr-defined]
        io.StringIO(tpp_as_xml))
    tpp_from_json = ThePhoenixProject.from_json(  # type:ignore[attr-defined]
        json.loads(tpp_as_json))
```

4.2 Logging and log access

This library utilizes an own instance of Logger, which you may access and add handlers to.

Listing 1: Example: send all logs messages to stdErr

```
import sys
import logging
import serializable

my_log_handler = logging.StreamHandler(sys.stderr)
my_log_handler.setLevel(logging.DEBUG)
my_log_handler.setFormatter(logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
↪%(message)s'))
```

(continues on next page)

(continued from previous page)

```
serializable.logger.addHandler(my_log_handler)
serializable.logger.setLevel(my_log_handler.level)
serializable.logger.propagate = False

@serializable.serializable_class
class Chapter:

    def __init__(self, *, number: int, title: str) -> None:
        self._number = number
        self._title = title

    @property
    def number(self) -> int:
        return self._number

    @property
    def title(self) -> str:
        return self._title


moby_dick_c1 = Chapter(number=1, title='Loomings')
print(moby_dick_c1.as_json())
```

CHAPTER

FIVE

SUPPORT

If you run into issues utilising this library, please raise a GitHub Issue. When raising an issue please include as much detail as possible including:

- Version of `py-serializable` you have installed
- Input(s)
- Expected Output(s)
- Actual Output(s)

5.1 Python Version Support

We endeavour to support all functionality for all current actively supported Python versions. However, some features may not be possible/present in older Python versions due to their lack of support - which are noted below.

CHAPTER

SIX

CHANGELOG

6.1 v0.17.1 (2024-01-07)

6.1.1 Fix

- Log placeholder (#60) (``3cc6cad`` <https://github.com/madpah/serializable/commit/3cc6cadad27a86b46ca576540f89a15f0f8fc1cd>`_)

6.1.2 Documentation

- Add "documentation" url to project meta (``bf864d7`` <https://github.com/madpah/serializable/commit/bf864d75d8a12426d4c71ae9ea1f533e730bd54e>`_)
- Add "documentation" url to project meta (``d3bcc42`` <https://github.com/madpah/serializable/commit/d3bcc4258ab8cdf6c9e09b47985997cafdc19e9a>`_)

6.2 v0.17.0 (2024-01-06)

6.2.1 Feature

- Logger (#47) (``9269b0e`` <https://github.com/madpah/serializable/commit/9269b0e681665abaef3f110925cd098b2438880f>`_)

6.2.2 Documentation

- Modernixe read-the-docs (``7ae6aad`` <https://github.com/madpah/serializable/commit/7ae6aad3b5939508238d1502c116866ef79949cb>`_)
- Homepage (#48) (``de206d6`` <https://github.com/madpah/serializable/commit/de206d6083be643a58f08554b61518367f67cda1>`_)
- Condaforge (#46) (``c0074ce`` <https://github.com/madpah/serializable/commit/c0074ce911f66bc6de0a451b8922f80f1ffa6270>`_)

6.3 v0.16.0 (2023-11-29)

6.3.1 Feature

- More controll over XML attribute serialization (#34) (``38f42d6` <https://github.com/madpah/serializable/commit/38f42d64e556a85206faa50459a9ce3e889bd3ae>`_)

6.4 v0.15.0 (2023-10-10)

6.4.1 Feature

- Allow custom (de)normalization (#32) (``aeecd6b` <https://github.com/madpah/serializable/commit/aeecd6b2e8c4e8febc84ebfa24fe7ec96fd9cb10>`_)

6.5 v0.14.1 (2023-10-08)

6.5.1 Fix

- JSON deserialize `Decimal` (#31) (``b6dc66a` <https://github.com/madpah/serializable/commit/b6dc66acfb7fdc82b3dd18caf4ad79ec0e87eef0>`_)

6.6 v0.14.0 (2023-10-06)

6.6.1 Feature

- Enhanced typehints and typing (#27) (``410372a` <https://github.com/madpah/serializable/commit/410372a0fa2713c5a36d790f08d2d4b52a6a187c>`_)

6.7 v0.13.1 (2023-10-06)

6.7.1 Fix

- Protect default value for `serialization_types` from unintended downstream modifications (#30) (``0e814f5` <https://github.com/madpah/serializable/commit/0e814f5248176e02a7f96480e54320dde781f8b2>`_)

6.7.2 Documentation

- Add examples to docs (#28) (``4eddb24`` <https://github.com/madpah/serializable/commit/4eddb242e51194694474748acdecd38b317b791e>`_)
- Remove unnecessary type-ignores (``26c561d`` <https://github.com/madpah/serializable/commit/26c561dc0bf9f5755899a8fa0d0a37aba6275074>`_)
- Remove unnecessary type-ignores (``11b5896`` <https://github.com/madpah/serializable/commit/11b5896057fd61838804ea5b52dc3bd0810f6c88>`_)

6.8 v0.13.0 (2023-10-01)

6.8.1 Feature

- Format specific (de)serialize (#25) (``dc998df`` <https://github.com/madpah/serializable/commit/dc998df37a2ba37fa43d10c8a1ce044a5b9f5b1e>`_)

6.9 v0.12.1 (2023-10-01)

6.9.1 Fix

- Xml defaultNamespace serialization and detection (#20) (``59eaa5f`` <https://github.com/madpah/serializable/commit/59eaa5f28eb2969e9d497669ef0436eb87b7525d>`_)

6.9.2 Documentation

- Render only public API (#19) (``fcc5d8e`` <https://github.com/madpah/serializable/commit/fcc5d8e6c49e8b8c199cb55f855d09e4259a075a>`_)
- Set codeblock language and caption (#15) (``5d5cf7b`` <https://github.com/madpah/serializable/commit/5d5cf7bc29ed70f4024c714b2326012a9db54cea>`_)

6.10 v0.12.0 (2023-03-07)

6.10.1 Feature

- Bump dev dependencies to latest (including `mypy` fixes) (``06dcaa2`` <https://github.com/madpah/serializable/commit/06dcaa28bfebb4505ddc67b287dc6f416822ffb6>`_)
- Bump dev dependencies to latest (including `mypy` fixes) (``6d70287`` <https://github.com/madpah/serializable/commit/6d70287640c411d33823e9188b0baa81fba80c24>`_)

6.11 v0.11.1 (2023-03-03)

6.11.1 Fix

- Use `defusedxml` whenever we load XML to prevent XEE attacks (``ae3d76c` <https://github.com/madpah/serializable/commit/ae3d76c31ab8af81d20acaaba45fd4bb9aad9305>`_)
- Use `defusedxml` whenever we load XML to prevent XEE attacks (``32fd5a6` <https://github.com/madpah/serializable/commit/32fd5a698b41b489b4643bcbe795e24a1e0db423>`_)
- Use `defusedxml` whenever we load XML to prevent XEE attacks (``72e0127` <https://github.com/madpah/serializable/commit/72e01279274246313170e5e7c9d32afec16edf7c>`_)
- Use `defusedxml` whenever we load XML to prevent XEE attacks (``de61deb` <https://github.com/madpah/serializable/commit/de61deb5c2447a656ca6a111194b2b0ceeab9278>`_)
- Use `defusedxml` whenever we load XML to prevent XEE attacks (``de26dc3` <https://github.com/madpah/serializable/commit/de26dc3d0eaab533dac9b1db40f0add56dd67754>`_)

6.12 v0.11.0 (2023-03-03)

6.12.1 Feature

- Disabled handling to avoid class attributes that clash with `keywords` and `builtins` (``4439227` <https://github.com/madpah/serializable/commit/44392274628ddec4aaaeae89a8387d435e3cf002>`_)

6.13 v0.10.1 (2023-03-02)

6.13.1 Fix

- Handle empty XML elements during deserialization (`f806f35` <https://github.com/madpah/serializable/commit/f806f3521f0afd8978f94f5ec355f47d9a538b91>`_)

6.14 0.10.0 (2023-02-21)

6.14.1 Feature

- Ability for custom `type_mapping` to take lower priority than `xml_array` (`fc0bb22` <https://github.com/madpah/serializable/commit/fc0bb22f395498be42394af5f70addb9f63f0b3a>`_)
- Handle `ForwardRef` types (`a66e700` <https://github.com/madpah/serializable/commit/a66e700eeb5a80447522b8112ecdeff0345f0608>`_)

6.15 v0.9.3 (2023-01-27)

6.15.1 Fix

- Deserializing JSON with custom JSON name was incorrect (``7d4aefc` <https://github.com/madpah/serializable/commit/7d4aefc98dfe39ae614227601369e9fd25c12faa>`_)

6.16 v0.9.2 (2023-01-27)

6.16.1 Fix

- Nested array of Enum values in `from_json()` failed (``ea4d76a` <https://github.com/madpah/serializable/commit/ea4d76a64c8c97f7cb0b16687f300c362dfe7623>`_)
- Output better errors when deserializing JSON and we hit errors (``1699c5b` <https://github.com/madpah/serializable/commit/1699c5b96bb6a8d4f034b29a6fe0521e3d650d53>`_)

6.17 v0.9.1 (2023-01-26)

6.17.1 Fix

- Nested array of Enum values in `from_xml()` failed (``393a425` <https://github.com/madpah/serializable/commit/393a4256abb69228a9e6c2fc76b508e370a39d93>`_)

6.18 v0.9.0 (2023-01-24)

6.18.1 Feature

- Bring library to BETA state (``c6c36d9` <https://github.com/madpah/serializable/commit/c6c36d911ae401af477bcc98633f10a87140d0a4>`_)

6.19 v0.8.2 (2023-01-23)

6.19.1 Fix

- Typing for `@serializable.view` was incorrect (``756032b` <https://github.com/madpah/serializable/commit/756032b543a2fedac1bb61f57796eea438c0f9a7>`_)
- Typing for `@serializable.serializable_enum` decorator was incorrect (``84e7826` <https://github.com/madpah/serializable/commit/84e78262276833f507d4e8a1ce11d4a82733f395>`_)

6.20 v0.8.1 (2023-01-23)

6.20.1 Fix

- Specific None value per View - support for XML was missing (``5742861` <https://github.com/madpah/serializable/commit/5742861728d1b371bc0a819fed0b12e9da5829e1>`_)

6.21 v0.8.0 (2023-01-20)

6.21.1 Feature

- Support for specific None values for Properties by View (``a80ee35` <https://github.com/madpah/serializable/commit/a80ee3551c5e23f9c0491f48c3f98022317ddd99>`_)

6.21.2 Fix

- Minor typing and styling (``b728c4c` <https://github.com/madpah/serializable/commit/b728c4c995076cd18317c878c6f5900c6b266425>`_)
- Minor typing and styling (``b2ebcfb` <https://github.com/madpah/serializable/commit/b2ebcfb53cd640eb70a51a9f637db24e0d7b367e>`_)

6.22 v0.7.3 (2022-09-22)

6.22.1 Fix

- None value for JSON is now `None` (`null`) (``8b7f973` <https://github.com/madpah/serializable/commit/8b7f973cd96c861c4490c50553c880e88ebf33dc>`_)

6.23 v0.7.2 (2022-09-22)

6.23.1 Fix

- Missing namespace for empty XML elements (``f3659ab` <https://github.com/madpah/serializable/commit/f3659ab9ea651dcd65168aa22fa838d35ee189d5>`_)

6.24 v0.7.1 (2022-09-15)

6.24.1 Fix

- Support forced inclusion of array properties by using `@serializable.include_none` (``7ad0ecf` <https://github.com/madpah/serializable/commit/7ad0ecf08c5f56de4584f4f081bfc0f667d2f477>`_)
- Support for deserializing to objects from a primitive value (``12f9f97` <https://github.com/madpah/serializable/commit/12f9f9711a5fd924898a0afb50a24c8d360ab3ff>`_)

6.25 v0.7.0 (2022-09-14)

6.25.1 Feature

- Support for including `None` values, restricted to certain Views as required (``614068a` <https://github.com/madpah/serializable/commit/614068a4955f99d8fce5da341a1fd74a6772b775>`_)

6.26 v0.6.0 (2022-09-14)

6.26.1 Feature

- Implement views for serialization to JSON and XML (``db57ef1` <https://github.com/madpah/serializable/commit/db57ef13fa89cc47db074bd9be4b48232842df07>`_)

6.26.2 Fix

- Support for `Decimal` in JSON serialization (``cc2c20f` <https://github.com/madpah/serializable/commit/cc2c20fe8bce46e4854cb0eecc6702459cd2f99a>`_)
- Better serialization to JSON (``e8b37f2` <https://github.com/madpah/serializable/commit/e8b37f2ee4246794c6c0e295bcdf32cd58d5e52d>`_)

6.27 v0.5.0 (2022-09-12)

6.27.1 Feature

- Support for string formatting of values (``99b8f3e` <https://github.com/madpah/serializable/commit/99b8f3e7ab84f087a87b330928fc598c96a0e682>`_)
- Support string formatting for values (``3fefe22` <https://github.com/madpah/serializable/commit/3fefe2294130b80f05e219bd655514a0956f7f93>`_)
- Support for custom Enum implementations (``c3622fc` <https://github.com/madpah/serializable/commit/c3622fcb0019de794b1cbd3ad6333b6044d8392a>`_)

6.28 v0.4.0 (2022-09-06)

6.28.1 Feature

- Add support for defining XML element ordering with `@serializable.xml_sequence()` decorator (``c1442ae` <https://github.com/madpah/serializable/commit/c1442aeb1776243922fbaa6b5174db5a54f71920>`_)

6.28.2 Fix

- Removed unused dependencies (``448a3c9` <https://github.com/madpah/serializable/commit/448a3c9f0de897cf1ee6d7c46af377c2f389730d>`_)
- Handle python builtins and keywords during `as_xml()` for element names (``3bbfb1b` <https://github.com/madpah/serializable/commit/3bbfb1b4a7808f4cedd3b2b15f31aaaf8e35d60a>`_)
- Handle python builtins and keywords during `as_xml()` for attributes (``8d6a96b` <https://github.com/madpah/serializable/commit/8d6a96b0850d4993c96cbc7d532d848ba9c5e8b3>`_)

6.29 v0.3.9 (2022-08-24)

6.29.1 Fix

- Support declaration of XML NS in `as_xml()` call (``19b2b70` <https://github.com/madpah/serializable/commit/19b2b7048fdd7048d62f618987c13f2d3a457726>`_)

6.30 v0.3.8 (2022-08-24)

6.30.1 Fix

- Deserialization of XML boolean values (``799d477` <https://github.com/madpah/serializable/commit/799d4773d858fdf8597bef905302a373ca150db8>`_)

6.31 v0.3.7 (2022-08-23)

6.31.1 Fix

- Fixed deferred parsing for Properties (``833e29b` <https://github.com/madpah/serializable/commit/833e29b8391c85931b12c98f87a2faf3a68d388e>`_)

6.32 v0.3.6 (2022-08-23)

6.32.1 Fix

- Support for cyclic dependencies (``911626c` <https://github.com/madpah/serializable/commit/911626c88fb260049fdf2931f6ea1b0b05d7166a>`_)

6.33 v0.3.5 (2022-08-22)

6.33.1 Fix

- Support for non-primitive types when XmlSerializationType == FLAT (``*7eff15b*`<https://github.com/madpah/serializable/commit/7eff15bbb8d20760418071c005d65d2623b44eab>`_)

6.34 v0.3.4 (2022-08-22)

6.34.1 Fix

- Support ENUM in XML Attributes (``*f2f0922*`<https://github.com/madpah/serializable/commit/f2f0922f2d0280185f6fc7f96408d6647588c8d2>`_)

6.35 v0.3.3 (2022-08-19)

6.35.1 Fix

- Handle Array types where the concrete type is quoted in its definition (``*b6db879*`<https://github.com/madpah/serializable/commit/b6db879d72822ada74a41362594b009f09349da9>`_)

6.36 v0.3.2 (2022-08-19)

6.36.1 Fix

- Work to support `sortedcontainers` as a return type for Properties (``*805a3f7*`<https://github.com/madpah/serializable/commit/805a3f7a10e41f63b132ac0bb234497d5d39fe2b>`_)

6.37 v0.3.1 (2022-08-19)

6.37.1 Fix

- Better support for Properties that have a Class type that is not a Serializable Class (e.g. UUID) (``*95d407b*`<https://github.com/madpah/serializable/commit/95d407b4456d8f106cf54ceb650cbde1aab69457>`_)

6.38 v0.3.0 (2022-08-19)

6.38.1 Feature

- Support ignoring elements/properties during deserialization (``6319d1f` <https://github.com/madpah/serializable/commit/6319d1f9e632a941b1d79a63083c1ecb194105be>`_)

6.39 v0.2.3 (2022-08-19)

6.39.1 Fix

- Update `helpers` to be properly typed (``d924dc2` <https://github.com/madpah/serializable/commit/d924dc2d3b5f02c61ff6ac36fa10fa6adaac7022>`_)

6.40 v0.2.2 (2022-08-19)

6.40.1 Fix

- Change to helpers to address typing issues (``1c32ba1` <https://github.com/madpah/serializable/commit/1c32ba143504a605a77df4908422a95d0bd07edf>`_)
- Remove `/` from method signature so we work on Python < 3.8 (``c45864c` <https://github.com/madpah/serializable/commit/c45864cd6c90ed38d8cedd944adcfe43b32326b2>`_)

6.41 v0.2.1 (2022-08-18)

6.41.1 Fix

- Update to work on python < 3.10 (``91df8cb` <https://github.com/madpah/serializable/commit/91df8cbb718db15ea182888aa796db32b8015004>`_)

6.42 v0.2.0 (2022-08-18)

6.42.1 Feature

- Library re-write to utilise decorators (``957fca7` <https://github.com/madpah/serializable/commit/957fca757d89dc1b8ef9b13357a5a9380dbe94ff>`_)

6.43 v0.1.7 (2022-08-15)

6.43.1 Fix

- Support for Objects that when represented in XML may just be simple elements with attributes (``*1369d7d*` <https://github.com/madpah/serializable/commit/1369d7d755d9e50273b72e2fdd7d2967442e5bde>`_)

6.44 v0.1.6 (2022-08-15)

6.44.1 Fix

- Temporarilty add `Any` as part of `AnySerializable` type (``*d3e9beb*` <https://github.com/madpah/serializable/commit/d3e9bebd7b8dc78d4eb36447ad0b1ee46e2745e0>`_)

6.45 v0.1.5 (2022-08-13)

6.45.1 Fix

- Direct support for Python `Enum` (``*50148cc*` <https://github.com/madpah/serializable/commit/50148cc98a26e4e51479b491acb58451ea5b42b6>`_)

6.46 v0.1.4 (2022-08-13)

6.46.1 Fix

- Added missing `py.typed` marker (``*ee3169f*` <https://github.com/madpah/serializable/commit/ee3169f466353a88922174b40f5b29cb98998be9>`_)

6.47 v0.1.3 (2022-08-12)

6.47.1 Fix

- Added helpers for serializing XML dates and times (xsd:date, xsd:datetime) (``*c309834*` <https://github.com/madpah/serializable/commit/c3098346abf445876d99ecb768d7a4a08b12a291>`_)

6.48 v0.1.2 (2022-08-12)

6.48.1 Fix

- Support for properties whose value is an `Type[SerializableObject]` but are not `List` or `Set` (``bf6773c`` <https://github.com/madpah/serializable/commit/bf6773c40f3f45dbe2821fdbe785b369f0b3b71c>`_)

6.49 v0.1.1 (2022-08-11)

6.49.1 Fix

- Handle nested objects that are not list or set (``4bc5252`` <https://github.com/madpah/serializable/commit/4bc525258d0ee655beabace18e41323b4b67ae1b>`_)

6.50 v0.1.0 (2022-08-10)

6.50.1 Feature

- First alpha release (``c95a772`` <https://github.com/madpah/serializable/commit/c95a7724186b6e45554624b5238c719d172ffc9f>`_)
- First working draft of library for (de-)serialization to/from JSON and XML (``7af4f9c`` <https://github.com/madpah/serializable/commit/7af4f9c4a100f1ce10502ecef228f42ea61e9c22>`_)

CHAPTER

SEVEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

7.1 serializable

7.1.1 Submodules

serializable.formatters

Module Contents

Classes

BaseNameFormatter	Helper class that provides a standard way to create an ABC using
CamelCasePropertyNameFormatter	Helper class that provides a standard way to create an ABC using
KebabCasePropertyNameFormatter	Helper class that provides a standard way to create an ABC using
SnakeCasePropertyNameFormatter	Helper class that provides a standard way to create an ABC using
CurrentFormatter	

class serializable.formatters.**BaseNameFormatter**

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

abstract classmethod encode(*property_name: str*) → str

abstract classmethod decode(*property_name: str*) → str

classmethod decode_as_class_name(*name: str*) → str

classmethod decode_handle_python_builtins_and_keywords(*name: str*) → str

[1] Created with sphinx-autoapi

classmethod encode_handle_python_builtins_and_keywords(*name: str*) → str

class serializable.formatters.**CamelCasePropertyNameFormatter**

Bases: *BaseNameFormatter*

Helper class that provides a standard way to create an ABC using inheritance.

classmethod encode(*property_name: str*) → str

classmethod decode(*property_name: str*) → str

classmethod decode_as_class_name(*name: str*) → str

classmethod decode_handle_python_builtins_and_keywords(*name: str*) → str

classmethod encode_handle_python_builtins_and_keywords(*name: str*) → str

class serializable.formatters.**KebabCasePropertyNameFormatter**

Bases: *BaseNameFormatter*

Helper class that provides a standard way to create an ABC using inheritance.

classmethod encode(*property_name: str*) → str

classmethod decode(*property_name: str*) → str

classmethod decode_as_class_name(*name: str*) → str

classmethod decode_handle_python_builtins_and_keywords(*name: str*) → str

classmethod encode_handle_python_builtins_and_keywords(*name: str*) → str

class serializable.formatters.**SnakeCasePropertyNameFormatter**

Bases: *BaseNameFormatter*

Helper class that provides a standard way to create an ABC using inheritance.

classmethod encode(*property_name: str*) → str

classmethod decode(*property_name: str*) → str

classmethod decode_as_class_name(*name: str*) → str

classmethod decode_handle_python_builtins_and_keywords(*name: str*) → str

classmethod encode_handle_python_builtins_and_keywords(*name: str*) → str

class serializable.formatters.**CurrentFormatter**

formatter: Type[*BaseNameFormatter*]

serializable.helpers

Module Contents

Classes

BaseHelper	Base Helper.
Iso8601Date	Base Helper.
XsdDate	Base Helper.
XsdDateTime	Base Helper.

class serializable.helpers.**BaseHelper**

Base Helper.

Inherit from this class and implement/override the needed functions!

This class does not provide any functionality, it is more like a Protocol with some fallback implementations.

abstract classmethod serialize(*o: Any*) → Any | str

general purpose serializer

abstract classmethod deserialize(*o: Any*) → Any

general purpose deserializer

classmethod json_normalize(*o: Any*, *, *view: Type[*serializable.ViewType*] | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any | None

json specific normalizer

classmethod json_serialize(*o: Any*) → str | Any

json specific serializer

classmethod json_denormalize(*o: Any*, *, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

json specific denormalizer

Parameters

- **tCls** – the class that was desired to denormalize to
- **pCls** – tha prent class - as context

classmethod json_deserialize(*o: Any*) → Any

json specific deserializer

classmethod xml_normalize(*o: Any*, *, *element_name: str*, *view: Type[*serializable.ViewType*] | None*, *xmlns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → xml.etree.ElementTree.Element | Any | None

xml specific normalizer

classmethod xml_serialize(*o: Any*) → str | Any

xml specific serializer

classmethod xml_denormalize(*o: xml.etree.ElementTree.Element*, *, *default_ns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

xml specific denormalizer

classmethod xml_deserialize(*o: str | Any*) → Any

xml specific deserializer

class serializable.helpers.**Iso8601Date**

Bases: *BaseHelper*

Base Helper.

Inherit from this class and implement/override the needed functions!

This class does not provide any functionality, it is more like a Protocol with some fallback implementations.

classmethod serialize(*o: Any*) → str

general purpose serializer

classmethod deserialize(*o: Any*) → datetime.date

general purpose deserializer

classmethod json_normalize(*o: Any*, *, *view: Type[*serializable.ViewType*] | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any | None

json specific normalizer

classmethod json_serialize(*o: Any*) → str | Any

json specific serializer

classmethod json_denormalize(*o: Any*, *, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

json specific denormalizer

Parameters

- **tCls** – the class that was desired to denormalize to
- **pCls** – tha prent class - as context

classmethod json_deserialize(*o: Any*) → Any

json specific deserializer

classmethod xml_normalize(*o: Any*, *, *element_name: str*, *view: Type[*serializable.ViewType*] | None*, *xmlns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → xml.etree.ElementTree.Element | Any | None

xml specific normalizer

classmethod xml_serialize(*o: Any*) → str | Any

xml specific serializer

classmethod xml_denormalize(*o: xml.etree.ElementTree.Element*, *, *default_ns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

xml specific denormalizer

classmethod xml_deserialize(*o: str | Any*) → Any

xml specific deserializer

class serializable.helpers.**XsdDate**

Bases: *BaseHelper*

Base Helper.

Inherit from this class and implement/override the needed functions!

This class does not provide any functionality, it is more like a Protocol with some fallback implementations.

classmethod serialize(*o: Any*) → str

general purpose serializer

classmethod deserialize(*o: Any*) → datetime.date

general purpose deserializer

classmethod json_normalize(*o: Any*, *, *view: Type[*serializable.ViewType*] | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any | None

json specific normalizer

classmethod json_serialize(*o: Any*) → str | Any

json specific serializer

classmethod json_denormalize(*o: Any*, *, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

json specific denormalizer

Parameters

- **tCls** – the class that was desired to denormalize to
- **pCls** – tha prent class - as context

classmethod json_deserialize(*o: Any*) → Any

json specific deserializer

classmethod xml_normalize(*o: Any*, *, *element_name: str*, *view: Type[*serializable.ViewType*] | None*, *xmlns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → xml.etree.ElementTree.Element | Any | None

xml specific normalizer

classmethod xml_serialize(*o: Any*) → str | Any

xml specific serializer

classmethod xml_denormalize(*o: xml.etree.ElementTree.Element*, *, *default_ns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

xml specific denormalizer

classmethod xml_deserialize(*o: str | Any*) → Any

xml specific deserializer

class serializable.helpers.**XsdDateTime**

Bases: *BaseHelper*

Base Helper.

Inherit from this class and implement/override the needed functions!

This class does not provide any functionality, it is more like a Protocol with some fallback implementations.

classmethod serialize(*o: Any*) → str

general purpose serializer

classmethod deserialize(*o: Any*) → datetime.datetime

general purpose deserializer

classmethod json_normalize(*o: Any*, *, *view: Type[*serializable.ViewType*] | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any | None

json specific normalizer

classmethod json_serialize(*o: Any*) → str | Any

json specific serializer

classmethod json_denormalize(*o: Any*, *, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

json specific denormalizer

Parameters

- **tCls** – the class that was desired to denormalize to
- **pCls** – tha prent class - as context

classmethod json_deserialize(*o: Any*) → Any

json specific deserializer

classmethod xml_normalize(*o: Any*, *, *element_name: str*, *view: Type[*serializable.ViewType*] | None*, *xmlns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → xml.etree.ElementTree.Element | Any | None

xml specific normalizer

classmethod xml_serialize(*o: Any*) → str | Any

xml specific serializer

classmethod xml_denormalize(*o: xml.etree.ElementTree.Element*, *, *default_ns: str | None*, *prop_info:* serializable.ObjectMetadataLibrary.SerializableProperty, *ctx: Type[Any]*, ***kwargs: Any*) → Any

xml specific denormalizer

classmethod xml_deserialize(*o: str | Any*) → Any

xml specific deserializer

7.1.2 Package Contents

Classes

ViewType	Base of all views.
SerializationType	Enum to define the different formats supported for serialization and deserialization.
XmlArraySerializationType	Enum to differentiate how array-type properties (think Iterables) are serialized.
ObjectMetadataLibrary	namespace-like

Functions

serializable_enum(...)	Decorator
serializable_class(...)	Decorator used to tell `serializable` that a class is to be included in (de-)serialization.
type_mapping(→ Callable[[_F], _F])	Decorator
include_none(→ Callable[[_F], _F])	Decorator
json_name(→ Callable[[_F], _F])	Decorator
string_format(→ Callable[[_F], _F])	Decorator
view(→ Callable[[_F], _F])	Decorator
xml_attribute(→ Callable[[_F], _F])	Decorator
xml_array(→ Callable[[_F], _F])	Decorator
xml_name(→ Callable[[_F], _F])	Decorator
xml_sequence(→ Callable[[_F], _F])	Decorator

Attributes

logger	The logger. The thing that captures all this package has to say.

serializable.**logger**

The logger. The thing that captures all this package has to say. Feel free to modify its level and attach handlers to it.

class serializable.**ViewType**

Base of all views.

class serializable.**SerializationType**

Bases: `str`, `enum.Enum`

Enum to define the different formats supported for serialization and deserialization.

JSON = 'JSON'

XML = 'XML'

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

`casefold()`

Return a version of the string suitable for caseless comparisons.

`center()`

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

`count()`

S.count(sub[, start[, end]]) -> int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

`encode()`

Encode the string using the codec registered for encoding.

encoding
The encoding in which to encode the string.

errors
The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

`endswith()`

S.endswith(suffix[, start[, end]]) -> bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

`expandtabs()`

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

`find()`

S.find(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

`format()`

S.format(*args, **kwargs) -> str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

`format_map()`

S.format_map(mapping) -> str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

`index()`

S.index(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

`isupper()`

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

`join()`

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

`ljust()`

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

`lower()`

Return a copy of the string converted to lowercase.

`lstrip()`

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

`partition()`

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

`removeprefix()`

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

`removesuffix()`

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

`replace()`

Return a copy with all occurrences of substring old replaced by new.

count
Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

`rfind()`

S.rfind(sub[, start[, end]]) -> int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

`rindex()`

S.rindex(sub[, start[, end]]) -> int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

`rjust()`

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

`rpartition()`

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

`rsplit()`

Return a list of the substrings in the string, using sep as the separator string.

sep
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit
Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

`rstrip()`

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

`split()`

Return a list of the substrings in the string, using sep as the separator string.

sep
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit
Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

`splitlines()`

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

`startswith`()

S.startswith(prefix[, start[, end]]) -> bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

`strip`()

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

`swapcase`()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

`title`()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

`translate`()

Replace each character in the string using the given translation table.

table
Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

`upper`()

Return a copy of the string converted to uppercase.

`zfill`()

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

`name`()

The name of the Enum member.

`value`()

The value of the Enum member.

class serializable.**XmlArraySerializationType**(**args*, ***kwds*)

Bases: `enum.Enum`

Enum to differentiate how array-type properties (think Iterables) are serialized.

Given a `Warehouse` has a property `boxes` that returns *List[Box]*:

`FLAT` would allow for XML looking like:

`` <warehouse>

<box>..box 1..</box> <box>..box 2..</box>

</warehouse> ``

`NESTED` would allow for XML looking like:

`` <warehouse>

<boxes>
<box>..box 1..</box> <box>..box 2..</box>

</boxes>

</warehouse> ``

FLAT = 1

NESTED = 2

name()

The name of the Enum member.

value()

The value of the Enum member.

class serializable.**ObjectMetadataLibrary**

namespace-like

The core Class in `serializable` that is used to record all metadata about classes that you annotate for serialization and deserialization.

class SerializableClass(*, *klass: type*, *custom_name: str | None = None*, *serialization_types: Iterable[SerializationType] | None = None*, *ignore_during_deserialization: Iterable[str] | None = None*)

Internal model class used to represent metadata we hold about Classes that are being included in (de-)serialization.

property name: str

property klass: type

property custom_name: str | None

property serialization_types: Iterable[*SerializationType*]

property ignore_during_deserialization: Set[str]

class SerializableProperty(*, *prop_name: str*, *prop_type: Any*, *custom_names: Dict[SerializationType, str]*, *custom_type: Any | None = None*, *include_none_config: Set[Tuple[Type[ViewType], Any]] | None = None*, *is_xml_attribute: bool = False*, *string_format_: str | None = None*, *views: Iterable[Type[ViewType]] | None = None*, *xml_array_config: Tuple[XmlArraySerializationType, str] | None = None*, *xml_sequence_: int | None = None*)

Internal model class used to represent metadata we hold about Properties that are being included in (de-)serialization.

property name: str

property custom_names: Dict[*SerializationType*, str]

property type_: Any

property concrete_type: Any

property custom_type: Any | None

property include_none: bool

property include_none_views: Set[Tuple[Type[*ViewType*], Any]]

property is_xml_attribute: bool

property string_format: str | None

property views: Set[Type[*ViewType*]]

property xml_array_config: Tuple[*XmlArraySerializationType*, str] | None

property is_array: bool

property is_enum: bool

property is_optional: bool

property xml_sequence: int

custom_name(*serialization_type:* SerializationType) → str | None

include_none_for_view(*view_: Type[*ViewType*]*) → bool

get_none_value_for_view(*view_: Type[*ViewType*] | None*) → Any

get_none_value(*view_: Type[*ViewType*] | None = None*) → Any

is_helper_type() → bool

is_primitive_type() → bool

parse_type_deferred() → None

custom_enum_klasses: Set[Type[enum.Enum]]

klass_mappings: Dict[str, *ObjectMetadataLibrary*]

klass_property_mappings: Dict[str, Dict[str, *ObjectMetadataLibrary*]]

classmethod defer_property_type_parsing(*prop:* ObjectMetadataLibrary, *klasses: Iterable[str]*) → None

classmethod is_klass_serializable(*klass: Any*) → bool

classmethod is_property(*o: Any*) → bool

classmethod register_enum(*klass: Type[_E]*) → Type[_E]

classmethod register_klass(*klass: Type[_T]*, *custom_name: str | None*, *serialization_types: Iterable[*SerializationType*]*, *ignore_during_deserialization: Iterable[str] | None = None*) → Type[_T] | Type[_JsonSerializable] | Type[_XmlSerializable]

classmethod register_custom_json_property_name(*qual_name: str*, *json_property_name: str*) → None

classmethod register_custom_string_format(*qual_name: str*, *string_format: str*) → None

classmethod register_custom_xml_property_name(*qual_name: str*, *xml_property_name: str*) → None

classmethod register_klass_view(*klass: Type[_T]*, *view_: Type[ViewType]*) → Type[_T]

classmethod register_property_include_none(*qual_name: str*, *view_: Type[ViewType] | None = None*, *none_value: Any | None = None*) → None

classmethod register_property_view(*qual_name: str*, *view_: Type[ViewType]*) → None

classmethod register_xml_property_array_config(*qual_name: str*, *array_type: XmlArraySerializationType*, *child_name: str*) → None

classmethod register_xml_property_attribute(*qual_name: str*) → None

classmethod register_xml_property_sequence(*qual_name: str*, *sequence: int*) → None

classmethod register_property_type_mapping(*qual_name: str*, *mapped_type: type*) → None

serializable.**serializable_enum**(*cls: Literal[None] = None*) → Callable[[Type[_E]], Type[_E]]

serializable.**serializable_enum**(*cls: Type[_E]*) → Type[_E]

Decorator

serializable.**serializable_class**(*cls: Literal[None] = None*, ***, *name: str | None = ...*, *serialization_types: Iterable[SerializationType] | None = ...*, *ignore_during_deserialization: Iterable[str] | None = ...*) → Callable[[Type[_T]], Type[_T] | Type[_JsonSerializable] | Type[_XmlSerializable]]

serializable.**serializable_class**(*cls: Type[_T]*, ***, *name: str | None = ...*, *serialization_types: Iterable[SerializationType] | None = ...*, *ignore_during_deserialization: Iterable[str] | None = ...*) → Type[_T] | Type[_JsonSerializable] | Type[_XmlSerializable]

Decorator used to tell `serializable` that a class is to be included in (de-)serialization.

Parameters

- **cls** – Class
- **name** – Alternative name to use for this Class
- **serialization_types** – Serialization Types that are to be supported for this class.
- **ignore_during_deserialization** – List of properties/elements to ignore during deserialization

Returns

serializable.**type_mapping**(*type_: type*) → Callable[[_F], _F]

Decorator

serializable.**include_none**(*view_: Type[ViewType] | None = None*, *none_value: Any | None = None*) → Callable[[_F], _F]

Decorator

serializable.**json_name**(*name: str*) → Callable[[_F], _F]

Decorator

serializable.**string_format**(*format_: str*) → Callable[[_F], _F]

Decorator

serializable.**view**(*view_: Type[ViewType]*) → Callable[[_F], _F]

Decorator

serializable.**xml_attribute**() → Callable[[_F], _F]

Decorator

serializable.**xml_array**(*array_type: XmlArraySerializationType*, *child_name: str*) → Callable[[_F], _F]

Decorator

serializable.**xml_name**(*name: str*) → Callable[[_F], _F]

Decorator

serializable.**xml_sequence**(*sequence: int*) → Callable[[_F], _F]

Decorator

PYTHON MODULE INDEX

S

INDEX

K

L

M

N

O

P

R

S

T

U

V

X

Z