jsontransform Documentation

Release 1.0.0-stable

Peter Morawski

Sep 15, 2018

Contents

json-transform is a small library to help you convert your python objects into JSON documents and vice versa. The source code of json-transform is hosted on Bitbucket.

New? Here is some help:

CHAPTER 1

Contents

1.1 Installation

1.1.1 PyPI

To install json-transform from PyPI you simply have to type the following command into the console

```
$ pip install json-transform
```

1.1.2 Git

To install the json-transform directly from the source which is hosted on Bitbucket you have to type the following commands into the console

```
$ git clone git@bitbucket.org:Peter-Morawski/json-transform.git
$ cd json-transform
$ python setup.py install
```

1.2 Getting Started

This guide will show you how you can start using json-transform in order to simplify your JSON object parsing.

1.2.1 What you'll learn

You'll learn how to define a JSON object in python using json-transform and how to encode your object instance into a JSON file as well as deserialize it from one.

1.2.2 Installing json-transform

Before you can start testing the following examples, you first need to install json-transform. To do this simply visit the *Installation* Page and follow the PyPI guide there.

1.2.3 Defining our first JSONObject using json-transform

Now that you have successfully installed json-transform we can finally start defining your first `JSONObject`. To do that you have to create a Plain Old Python Object. It can have any amount of methods, properties, etc... The important part is

1. it needs to extend the `JSONObject` class so that json-transform will recognize that this object is intended to be a encodable and decodable to a JSON document.
2. it needs to have at least one property decorated with the `field()` decorator.

So let's define a simple entity.

```
from jsontransform import JSONObject, field


class Person(JSONObject):
    def __init__(self):
        self._first_name = ""
        self._last_name = ""

    @property
    @field("firstName", required=True)
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, value):
        self._first_name = value

    @property
    @field("lastName")
    def last_name(self):
        return self._last_name

    @last_name.setter
    def last_name(self, value):
        self._last_name = value
```

In this example we have given the *first_name* and the *last_name* property a custom **field_name** so when we encode our `JSONObject` the fields in the resulting JSON document will be called **firstName** and **lastName**. The same applies for the decoding. The decoder will search for fields called *firstName* and *lastName*. We will see this later in action.

Besides a **field_name** the *first_name* property has the **required** parameter set to True. This means that this `field()` is mandatory when we want to decode a JSON document into our `JSONObject`.

Now that we have defined our entity let's create an instance of it.

```
peter = Person()
peter.first_name = "Peter"
peter.last_name = "Parker"
```

1.2.4 Encoding

When we want to encode our *JSONObject* we can use the following functions

- *dump()* to encode it into a *write()* supporting file-like object
- *dumps()* to encode it into an *str* or
- *dumpd()* to encode it into a *dict*

It is also possible to encode our *JSONObject* using the *JSONEncoder* but to keep it simple we will use the *dumpd()* function to encode our *JSONObject* into a *dict*.

To keep things simple we will use the *dumpd()* function to encode our *JSONObject* into a dict which is JSON conform.

```
from jsontransform import dumpd

dumpd(peter)
# result: {'age': 56, 'birthday': '1962-09-23', 'firstName': 'Peter', 'lastName':
↪'Parker'}
```

1.2.5 Decoding

When we want to decode a file, *dict* or an *str* into our *JSONObject* we can use the following functions

- *load()* to decode a *JSONObject* from a *read()* supporting file-like object
- *loads()* to decode a *JSONObject* from an *str* or
- *loadd()* to decode a *JSONObject* from a *dict*

We also have a *JSONDecoder* which can be instantiated and provides the same functionality like the previously mentioned functions but to keep it simple we'll use the *loadd()* function to decode a *dict* into our *JSONObject*.

```
from jsontransform import loadd

peter = loadd({'age': 56, 'birthday': '1962-09-23', 'firstName': 'Peter', 'lastName':
↪'Parker'})

print(type(peter))
# result <class 'Person'>

print(peter.first_name)
# result: Peter

print(peter.last_name)
# result: Parker
```

Note: When decoding into a *JSONObject* we can specify the target type / the *JSONObject* into which the JSON document should be decoded OR we can let json-transform find the most matching *JSONObject* by itself *(like in the example above)*.

After the decoding our fields/properties will be casted into their appropriate type. To see which types are supported check the *Fields* page.

1.3 Fields

1.3.1 Encoding

Supported field types

- *None*
- *str*
- *unicode*
- *int*
- *float*
- *list*
- *tuple*
- *dict*
- *set*
- `datetime.date`
- `datetime.datetime` (with timezone or without)
- `JSONObject`

Note: Types like *set*, *tuple* will be converted into a list during the serialization process and can't be decoded into their original types.

1.3.2 Decoding

Python Type	JSON Type	JSON Example Value
None	null	
str	string	
unicode	string	
int	number	12
float	number	12.24
list	array	
set	array	
tuple	array	
dict	object	{"firstName": ""}
`datetime.date`	str	"2018-08-06" (ISO 8601 formatted date)
`datetime.datetime`	str	"2018-08-06T18:00:00Z" / "2018-08-06T18:00:00+0100" (ISO 8601 formatted datetime)

1.4 API

exception jsontransform.**ConfigurationError**
The passed *JSONObject* was not configured correctly.

exception jsontransform.**ConstraintViolationError**
A constraint which has been defined on a *field()* has been violated.

class jsontransform.**FieldMode**
The *FieldMode* describes the behavior of the *field()* during the encoding/decoding process. It marks that the *field()* should not be in the JSON document when the *JSONObject* is encoded but it should be decoded and vice versa.

DECODE = 'd'
Indicates that the *field()* can **ONLY** be decoded.

ENCODE = 'e'
Indicates that the *field()* can **ONLY** be encoded.

ENCODE_DECODE = 'ed'
Indicates that the *field()* can be encoded **AND** decoded.

class jsontransform.**JSONDecoder**
This class offers methods to decode a JSON document into a *JSONObject*. A *JSONObject* can be decoded from

- an *str*
- a *dict*
- a *write()* supporting file-like object

from_json_dict(*json_dict*, *target=None*)
Decode a python *dict* into a *JSONObject*. The *dict* **MUST** be JSON conform so it cannot contain other object instances.

Parameters

- **json_dict** – The dict which should be decoded
- **target** – (optional) The type of the target JSONObject into which this dict should be decoded. When this is empty then the target JSONObject will be searched automatically

Raises

- ***ConfigurationError*** – When the target JSONObject does NOT define any JSON fields
- **TypeError** – When the signature of the passed target did NOT match the signature of the passed dict i.e. they had no fields in common
- ***MissingObjectError*** – When no target JSONObject was specified AND no matching JSONObject could be found
- ***ConstraintViolationError*** – When a field inside the dict violated a constraint which is defined on the target JSONObject e.g. a required field is missing

Returns A JSONObject which matched the signature of the dict and with the values of it

from_json_file(*json_file*, *target=None*)
Decode a *read()* supporting file-like object into a *JSONObject*. The file-like object **MUST** contain a valid JSON document.

Parameters

- **json_file** – The read() supporting file-like object which should be decoded into a JSONObject
- **target** – (optional) The type of the target JSONObject into which this file-like object should be decoded. When this is empty then the target JSONObject will be searched automatically

Raises

- ***ConfigurationError*** – When the target JSONObject does NOT define any JSON fields
- **TypeError** – When the signature of the passed target did NOT match the signature of the JSON document which was read from the passed file-like object i.e. they had no fields in common
- ***MissingObjectError*** – When no target JSONObject was specified AND no matching JSONObject could be found
- ***ConstraintViolationError*** – When a field of the JSON document which was read from the file-like object violated a constraint which is defined on the target JSONObject e.g. a required field is missing

Returns A JSONObject which matched the signature of the JSON document which the read() supporting file-like object returned and with the values of it

from_json_str (*json_str*, *target=None*)

Decode an *str* into a *JSONObject*. The *str* **MUST** contain a JSON document.

Parameters

- **json_str** – The str which should be decoded
- **target** – (optional) The type of the target JSONObject into which this str should be decoded. When this is empty then the target JSONObject will be searched automatically

Raises

- ***ConfigurationError*** – When the target JSONObject does NOT define any JSON fields
- **TypeError** – When the signature of the passed target did NOT match the signature of the JSON document which was inside the passed str i.e. they had no fields in common
- ***MissingObjectError*** – When no target JSONObject was specified AND no matching JSONObject could be found
- ***ConstraintViolationError*** – When a field of the JSON document which was inside the str violated a constraint which is defined on the target JSONObject e.g. a required field is missing

Returns A JSONObject which matched the signature of the JSON document from the str and with the values of it

static validate_required_fields (*json_object*, *json_dict*)

Validate if a *dict* which will be decoded satisfied all required fields of the *JSONObject* into which it will be decoded.

Parameters

- **json_object** – The instance of the JSONObject into which the dict will be decoded
- **json_dict** – The dict which should be validated

Raises **ConstraintValidationError** – When a required field is missing

class jsontransform.**JSONEncoder**

This class offers methods to encode a *JSONObject* into JSON document. A *JSONObject* can be encoded to

- an *str*
- a *dict*
- a *write()* supporting file-like object

to_json_dict(*json_object*)

Encode an instance of a *JSONObject* into a python *dict*.

Parameters **json_object** – The instance of the JSONObject which should be encoded

Raises

- ***ConfigurationError*** – When the JSONObject of which an instance was passed does NOT define any JSON fields
- **TypeError** – When the type of a field in the JSONObject is not encodable

Returns A dict which represents the passed JSONObject and is JSON conform

to_json_file(*json_object*, *json_file*)

Encode an instance of a *JSONObject* and write the result into a *write()* supporting file-like object.

Parameters

- **json_object** – The instance of the JSONObject which should be encoded
- **json_file** – A write() supporting file-like object

Raises

- ***ConfigurationError*** – When the JSONObject of which an instance was passed does NOT define any JSON fields
- **TypeError** – When the type of a field in the JSONObject is not encodable

to_json_str(*json_object*)

Encode an instance of a *JSONObject* into an *str* which contains a JSON document.

Parameters **json_object** – The instance of the JSONObject which should be encoded

Raises

- ***ConfigurationError*** – When the JSONObject of which an instance was passed does NOT define any JSON fields
- **TypeError** – When the type of a field in the JSONObject is not encodable

Returns An str which contains the JSON representation of the passed JSONObject

class jsontransform.**JSONObject**

Every entity/class which is intended to be encodable and decodable to a JSON document **MUST** inherit/extend this class.

exception jsontransform.**MissingObjectError**

No *JSONObject* which matches the signature of the passed JSON document could be found.

jsontransform.**dump**(*json_object*, *json_file*)

Shortcut for instantiating a new *JSONEncoder* and calling the `to_json_file()` function.

See also:

For more information you can look at the doc of *JSONEncoder.to_json_file()*.

jsontransform.**dumpd**(*json_object*)
Shortcut for instantiating a new *JSONEncoder* and calling the `to_json_dict()` function.

See also:

For more information you can look at the doc of *JSONEncoder.to_json_dict()*.

jsontransform.**dumps**(*json_object*)
Shortcut for instantiating a new *JSONEncoder* and calling the `to_json_str()` function.

See also:

For more information you can look at the doc of *JSONEncoder.to_json_str()*.

jsontransform.**field**(*field_name=None*, *required=False*, *mode='ed'*, *func=None*)
The *field()* decorator is used to mark that a `property` inside a *JSONObject* is a JSON field so it will appear in the JSON document when the *JSONObject* is encoded or decoded.

Note:

- The brackets *()* after the @field decorator are important even when no additional arguments are given
- The `property` decorator must be at the top or else the function won't be recognized as a property

---

Parameters

- **func** – The method which is decorated with @property decorator.
- **field_name** – (optional) A name/alias for the field (how it should appear in the JSON document) since by default the name of the property will be used.
- **required** – (optional) A *bool* which indicates if this field is mandatory for the decoding process. When a field which is marked as required does NOT exist in the JSON document from which the JSONObject is decoded from, a ConstraintViolationError will be raised. (False by default)
- **mode** – (optional) The FieldMode of the field. (ENCODE_DECODE by default)

jsontransform.**load**(*json_file*, *target=None*)
Shortcut for instantiating a new *JSONDecoder* and calling the `from_json_file()` function.

See also:

For more information you can look at the doc of *JSONDecoder.from_json_file()*.

jsontransform.**loadd**(*json_dict*, *target=None*)
Shortcut for instantiating a new *JSONDecoder* and calling the `from_json_dict()` function.

See also:

For more information you can look at the doc of *JSONDecoder.from_json_dict()*.

jsontransform.**loads**(*json_str*, *target=None*)
Shortcut for instantiating a new *JSONDecoder* and calling the `from_json_str()` function.

See also:

For more information you can look at the doc of *JSONDecoder.from_json_str()*.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index