DoJSON Documentation

Release 1.1.0

Invenio collaboration

March 11, 2016

Contents

CHAPTER 1

About

DoJSON is a simple Pythonic JSON to JSON converter.

CHAPTER 2

Installation

DoJSON is on PyPI so all you need is:

```
$ pip install dojson
```

CHAPTER 3

Documentation

Documentation is readable at https://pythonhosted.org/dojson/ or it can be built using Sphinx:

```
$ pip install dojson[docs]
$ python setup.py build_sphinx
```

CHAPTER 4

Testing

Running the test suite is as simple as:

```
$ python setup.py test
```

CHAPTER 5

Example

A simple example on how to convert MARCXML to JSON:

```
from dojson.contrib.marc21.utils import create_record, split_stream
from dojson.contrib.marc21 import marc21
[marc21.do(create_record(data)) for data in split_stream(open('/tmp/data.xml', 'r'))]
```

5.1 User's Guide

This part of the documentation will show you how to get started in using DoJSON.

5.1.1 Usage

DoJSON is a simple Pythonic JSON to JSON converter.

The main goal of this package is to help with managing a set of rules for manipulation of Python dictionaries with focus on JSON serialization. Each rule is associated with regular expression and key. The regular expression has to match a key in the source mapping and produces a new value that is added to the output mapping under the new key.

Initialization

First create an *Overdo* object that is holding the index with rules.

```
>>> import dojson
>>> simple = dojson.Overdo()
```

Next step is to create rules that will manupulate a source object.

```
>>> @simple.over('first', '^.*st$')
... def first(self, key, value):
...     return value + 1
>>> @simple.over('second', '^.*nd$')
... def second(self, key, value):
...     return value + 2
```

And now we can try to match the source object and produce new data.

```
>>> data = simple.do({'1st': 1, '2nd': 2})
>>> assert 2 == data['first']
>>> assert 4 == data['second']
```

Command line interface

Command line interface script is installed as `dojson`.

The easiest way to get started by applying already registered rule to a JSON data.

```
{"245__": {"a": "Test title"}}
```

DoJSON comes with set of rules for processing MARC21 fields.

```
$ echo '{"245__": {"a": "Test title"}}' | dojson do marc21
{"title_statement": {"title": "Test title"}}
```

Sometimes one can get input with fields that does not match any rule. To get such a list of fields one can use the `missing` command.

```
$ echo '{"999__": {"a": "Test title"}}' | dojson missing marc21
999__
```

The usual problem comes with reading different file formats such as XML.

```
<?xml version='1.0' encoding='UTF-8'?>
<collection xmlns="http://www.loc.gov/MARC21/slim">
  <record>
    <datafield tag="245" ind1=" " ind2=" ">
      <subfield code="a">Test title</subfield>
    </datafield>
  </record>
</collection>
```

You can specify regitered loader using `-l <NAME>` argument. Save the above example as `example.xml` and check following command.

```
$ dojson -i example.xml -l marcxml do marc21
{"title_statement": {"title": "Test title"}}
```

In similar way it is possible to specify different output serializer (`-d`).

```
$ echo '{"title_statement": {"title": "Test title"}}' | \
  dojson -d marcxml do marc21
<?xml version='1.0' encoding='UTF-8'?>
<collection xmlns="http://www.loc.gov/MARC21/slim">
  <record>
    <datafield tag="245" ind1=" " ind2=" ">
      <subfield code="a">Test title</subfield>
    </datafield>
  </record>
</collection>
```

Command chaining

This makes JSON manipulation even easier. For first example see `schema` command that accept string argument containing URL of JSON-Schema that should be added to `$schema` field.

```
$ dojson -i example.xml -l marcxml do marc21 \
  schema http://example.org/schema/marc21.json
..."schema": "http://example.org/schema/marc21.json"...
```

Second example shows easy verification that rules produce an identity function.

```
$ dojson -l marcxml -d marcxml do marc21 do to_marc21 < example.xml | \
  diff - example.xml
```

Extensibility

New commands, loaders, dumpers, or rules can be provided via entry points.

- `dojson.cli` commands that return a processor acception an iterator;
- `dojson.cli.load` functions expecting a stream and returning Python dict or iterator;
- `dojson.cli.dump` functions expecting a Python object and returning `str`;
- `dojson.cli.rule` instances of *`dojson.overdo.Overdo`* with loaded rules.

5.2 API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

5.2.1 API

Do JSON translation.

class `dojson.overdo.`**`Index`**(*rules=None*, *flags=0*, *branch_size=99*)
Index implementation based on build-in Python SRE module.

`query`(*key*)
Return data matching the key.

class `dojson.overdo.`**`Overdo`**(*bases=None*, *entry_point_group=None*)
Translation index.

`build`()
Build.

`do`(*blob*, *ignore_missing=True*, *exception_handlers=None*)
Translate blob values and instantiate new model instance.

Raises `MissingRule` when no rule matched and `ignore_missing` is `False`.

Parameters

- **`blob`** – `dict`-like object on which the matching rules are going to be applied.
- **`ignore_missing`** – Set to `False` if you prefer to raise an exception `MissingRule` for the first key that it is not matching any rule.
- **`exception_handlers`** – Give custom exception handlers to take care of non-standard codes that are installation specific.

New in version 1.0.0: `ignore_missing` allows to specify if the function should raise an exception.

New in version 1.1.0: `exception_handlers` allows to set custom handlers for non-standard MARC codes.

`missing`(*blob*)
Return keys with missing rules.

over(*name*, **source_tags*)
Register creator rule.

CLI

Define chainable commands for processing loaded data.

dojson.cli.command.**process_do** = <click.core.Command object>
Process data using given rule.

dojson.cli.command.**process_missing** = <click.core.Command object>
List fields with missing rules.

dojson.cli.command.**process_schema** = <click.core.Command object>
Add $schema to an item.

Utility function to manage CLI entry points

dojson.cli.utils.**open_entry_point**(*group_name*)
Open entry point.

dojson.cli.utils.**with_plugins**(*group_name*)
Register external CLI commands.

Contrib

There are set of rules to manage translation from other formats.

MARC21

MARC standards based on www.loc.gov/marc/.

5.3 Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

5.3.1 Contributing

Bug reports, feature requests, and other contributions are welcome. If you find a demonstrable problem that is caused by the code of this library, please:

1. Search for already reported problems.
2. Check if the issue has been fixed or is still reproducible on the latest *master* branch.
3. Create an issue with **a test case**.

If you create a feature branch, you can run the tests to ensure everything is operating correctly:

```
$ python setup.py test

...

====== 31 passed, 23 skipped in 1.37 seconds ======
```

You can also test your feature branch using Docker:

```
$ docker-compose build
$ docker-compose run web python setup.py test
$ docker-compose run web python setup.py build_sphinx
$ docker-compose run web pep257 --match-dir='dojson'
```

5.3.2 Changes

Version 1.1.0 (released 2016-03-10):

Incompatible changes

- Moves *–load* and *–dump* options to global group.

New features

- Adds *schema* command to enhance JSON with '$schema' field. (#73)
- Adds rules and schemas for MARC 21 Format for Authority Data. (#7)
- Adds rules and schemas for MARC 21 Format for Holdings Data. (#21)
- Adds support for parsing <*leader/*> tag in MARCXML.
- Adds new parameter *exception_handlers* to dojson.Overdo.do and dojson.contrib.to_marc21.model.Underdo.do. It can be given to the translation process to deal with non-standard fields in a custom way (#26).
- Adds new utility *map_order* function to ease renaming of subfields.

Improved features

- Adds more detailed usage examples. (#117)
- Refactors CLI to allow commands chaining.
- Adds support preserving the order of subfields.

Bug fixes

- Fixes support for Python 3.5.1.

Version 1.0.0 (released 2016-01-14):

Incompatible changes

- Removes support for single key matching multiple rules. Please make your rules mutually exclusive!
- controlfields 00x are expected to be the element or a list of multiple elements.

New features

- Adds new keyword argument *ignore_missing* to *Overdo.do* method to specify if method should raise *MissingRule* exception when there is no matching rule for a key.
- Adds new CLI option *–strict* to the *do* command that sets the *ignore_missing* argument to *False*. (#51)
- MARC XML serialization from to_marc21.

Improved features

- Adds support for Python 3+.
- Uses an OrderedDict to let the external tools working on *dict* (like json) behave correctly.
- All results from rules using *for_each_value* decorator are being automatically extended. This is useful for repeatable MARC21 fields with different indicators. (#53)
- Record are stored in an immutable sorted structure which enables to keep the intended order while offering easy ways to access, index and manipulate.
- Adds two records to be tested.
- Reorders some of the assertion: *expected* == *actual*.

Version 0.4.0 (released 2015-11-18):

New features

- Improves dojson.contrib.marc2.utils.load() to read the input by iterating of the open stream, rather than loading it all in memory in one go. (#45) (#46)
- Renames OverUndo to Underdo following same name convention as for Overdo.

Bug fixes

- Fixes indicator extraction from value in *Underdo* model.

Version 0.3.0 (released 2015-11-09):

New features

- Adds **experimental** rules for converting human readable JSON into a JSON representation of the MARC21 Format.
- Adds *do* and *missing* commands for *dojson* command line interface (see *dojson –help* for more information).

Improved features

- Adds missing mapping for the first indicator of field 856.

Version 0.2.0 (released 2015-10-07):

New features

- Adds the posibility to use base DoJSON model so the rules are "inherited" from them.
- Adds new decorator *ignore_value* that remove the key in the resulting json for None value.

Improved features

- Uses entry points instead of plain imports to load the creator rules.

Bug fixes

- Removes calls to PluginManager consider_setuptools_entrypoints() removed in PyTest 2.8.0.

Version 0.1.1 (released 2015-07-27):

- Sorts and removes duplicated enum values.
- Swaps wrongly defined repeatable and non-repeatable subfields. (#23)
- Addresses issue when allowed indicators where defined as a range. (#22)

Version 0.1.0 (released 2015-07-03):

- Initial public release.

5.3.3 License

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.

In applying this license, CERN does not waive the privileges and
immunities granted to it by virtue of its status as an
Intergovernmental Organization or submit itself to any jurisdiction.
```

5.3.4 Authors

DoJSON is developed for the Invenio digital library software.

Contact us at info@invenio-software.org.

Active contributors:

- Jiri Kuncar <jiri.kuncar@cern.ch>
- Esteban J. G. Gabancho <esteban.jose.garcia.gabancho@cern.ch>
- Sami Hiltunen <sami.mikael.hiltunen@cern.ch>
- Tibor Simko <tibor.simko@cern.ch>

d

B

D

I

M

O

P

Q

W