dublincore Documentation

Release 0.1.1

CERN

Mar 21, 2018

Contents

Dublin Core XML generation from Python dictionaries.

- Free software: MIT license
- Documentation: https://dcxml.readthedocs.io/

CHAPTER 1

User's Guide

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

1.1 Installation

Dublin Core package is on PyPI so all you need is:

```
$ pip install dcxml
```

1.2 Usage

Dublin Core XML generation from Python dictionaries.

The Dublin Core Python package allows you to generate XML (either as string or ElementTree) for the 15 elements in Simplified Dublin Core. By default the package wraps the 15 elements in an OAI DC element (customizable).

See http://dublincore.org/documents/dces/ for the description of all 15 elements.

First let's create a Python dictionary with all 15 elements:

```
>>> data = dict(
...     contributors = ['CERN'],
...     coverage = ['Geneva'],
...     creators = ['CERN'],
...     dates = ['2002'],
...     descriptions = ['Simple Dublin Core generation'],
...     formats = ['application/xml'],
...     identifiers = ['dublin-core'],
...     languages = ['en'],
...     publishers = ['CERN'],
...     relations = ['Invenio Software'],
...     rights = ['MIT'],
```

```
...     sources = ['Python'],
...     subject = ['XML'],
...     titles = ['Dublin Core XML'],
...     types = ['Software'],
... )
```

The structure of the dictionary is:

- Keys: Normally the plural version of the DC element name except for (`rights` and `coverage`).
- Values: List of strings. Each item in the list will result in one DC element.

Now, let's generate the Dublin Core XML:

```
>>> from dcxml import simpledc
>>> xml = simpledc.tostring(data)
```

and print the 15 elements (without the container element)

```
>>> for l in xml.splitlines()[2:-1]:
...     print(l)
  <dc:contributor>CERN</dc:contributor>
  <dc:coverage>Geneva</dc:coverage>
  <dc:creator>CERN</dc:creator>
  <dc:date>2002</dc:date>
  <dc:description>Simple Dublin Core generation</dc:description>
  <dc:format>application/xml</dc:format>
  <dc:identifier>dublin-core</dc:identifier>
  <dc:language>en</dc:language>
  <dc:publisher>CERN</dc:publisher>
  <dc:relation>Invenio Software</dc:relation>
  <dc:rights>MIT</dc:rights>
  <dc:source>Python</dc:source>
  <dc:title>Dublin Core XML</dc:title>
  <dc:type>Software</dc:type>
```

The container element is by default the `<oai_dc:dc>` element:

```
>>> print(xml.splitlines()[1])
<oai_dc:dc ...
```

In case you need an ElementTree instead of a string, it's as simple as:

```
>>> tree = simpledc.dump_etree(data)
```

CHAPTER 2

API Reference

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

2.1 Simple Dublin Core

Generation of Simple Dublin Core XML v1.1.

By default the package will wrap elements in an OAI DC element. This behavior can be changed by specifying `container`, `nsmap` and `attribs` to the API functions.

`dcxml.simpledc.`**`dump_etree`**(*data*, *container=None*, *nsmap=None*, *attribs=None*)

Convert dictionary to Simple Dublin Core XML as ElementTree.

Parameters

- **`data`** – Dictionary.
- **`container`** – Name (include namespace) of container element.
- **`nsmap`** – Namespace mapping for lxml.
- **`attribs`** – Default attributes for container element.

Returns LXML ElementTree.

`dcxml.simpledc.`**`tostring`**(*data*, ***kwargs*)

Convert dictionary to Simple Dublin Core XML as string.

Parameters

- **`data`** – Dictionary.
- **`container`** – Name (include namespace) of container element.
- **`nsmap`** – Namespace mapping for lxml.
- **`attribs`** – Default attributes for container element.

Returns LXML ElementTree.

dcxml.simpledc.**ns = {'xml': 'xml', 'oai_dc': 'http://www.openarchives.org/OAI/2.0/oai_dc/**
Default namespace mapping.

dcxml.simpledc.**container_attribs = {'{http://www.w3.org/2001/XMLSchema-instance}schemaLocat**
Default container element attributes.

dcxml.simpledc.**container_element = '{http://www.openarchives.org/OAI/2.0/oai_dc/}dc'**
Default container element.

CHAPTER 3

Additional Notes

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

3.1 Contributing

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

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/dublincore/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

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

Implement Features

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

Write Documentation

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

Submit Feedback

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

If you are proposing a feature:

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

3.1.2 Get Started!

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

1. Fork the *invenio* repo on GitHub.
2. Clone your fork locally:

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

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv dcxml
$ cd dcxml/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

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

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

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

```
$ git add .
$ git commit -s -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

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

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.org/inveniosoftware/dcxml/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 0.1.0 (released 2016-03-21)

- Initial public release.

3.3 License

3.4 Contributors

- Jiri Kuncar
- Lars Holm Nielsen
- Tibor Simko

Python Module Index

d

Index