Python-OOXML Documentation

Release 0.1

Aleksandar Erkalovic

Jun 26, 2017

Contents

1 Python-OOXML **3**
- 1.1 Installation . . . 3
- 1.2 Usage . . . 3
- 1.3 Extending . . . 4

2 API **7**
- 2.1 Python-OOXML API . . . 7

3 Indices and tables **9**

Python-OOXML is a Python library for parsing Office Open XML files. At the moment it only supports HTML as output format. Strong emphasis is put on easy customization of the output. The library comes with an importer which is capable of splitting a document into separate chapters. It works both with documents which use Word styles, and documents where they are not used.

Homepage: https://github.com/booktype/python-ooxml

Python-OOXML is used in Booktype 2.0 from Sourcefabric.

CHAPTER 1

Python-OOXML

Installation

Install with pip

```
$ pip install python-ooxml
```

Install from the source

```
$ git clone https://github.com/booktype/python-ooxml.git
$ python build.py install
$ pip install -r requirements/base.txt
```

Usage

Parsing

```
import sys
import logging

from lxml import etree

import ooxml
from ooxml import parse, serialize, importer

logging.basicConfig(filename='ooxml.log', level=logging.INFO)

if len(sys.argv) > 1:
```

```
    file_name = sys.argv[1]

    dfile = ooxml.read_from_file(file_name)

    print serialize.serialize(dfile.document)
    print serialize.serialize_styles(dfile.document)
```

Extending

Serializer

Serializer is used to generate ElementTree node for different elements we have already parsed. Serializers work with ElementTree API because we want to be able to easily manipulate with our generated content in serializers ands hooks. Generated tree is converted to HTML textual representation at the end of the process.

Serializer is passed reference to element where new content should be inserted. When serializer is done it calls hooks defined for this kind of element.

Supported OOXML document elements are:

- Paragraph
- Text
- Link
- Image
- Table / Table Cell
- Footnote
- Symbol
- List
- Break
- Table Of Contents (just parsed)
- TextBox (just parsed)
- Math (just parsed)

```
import ooxml
from ooxml import serialize

def serialize_break(ctx, document, elem, root):
    if elem.break_type == u'textWrapping':
        _div = etree.SubElement(root, 'br')
    else:
        _div = etree.SubElement(root, 'span')
        _div.set('style', 'page-break-after: always;')

    serialize.fire_hooks(ctx, document, elem, _div, ctx.get_hook('page_break'))

    return root


dfile = ooxml.read_from_file('doc_with_math_element.docx')
```

```
opts = {
  'serializers': {
     doc.Break: serialize_break,
    }
}

print serialize.serialize(dfile.document, opts)
```

Hook

Hooks are used for easy and quick manipulation with generated ElementTree elements. Hooks are called for each newly created element. Using hooks we are able to slightly modify or completely rewrite content generated by serializers.

Example

We are using MS Word to edit our document. Using style "Quote" we mark certain parts of our document as quote and using style "Title" we marked the title. Sample code which uses hooks will put the title inside of <h1> element and add class "our_quote" to the quote element.

Sample code

```
import six

import ooxml
from ooxml import parse, serialize, importer

def check_for_header(ctx, document, el, elem):
    if hasattr(el, 'style_id'):
        if el.style_id == 'Title':
            elem.tag = 'h1'

def check_for_quote(ctx, document, el, elem):
    if hasattr(el, 'style_id'):
        if el.style_id == 'Quote':
            elem.set('class', elem.get('class', '') + ' our_quote')

file_name = '../files/03_hooks.docx'
dfile = ooxml.read_from_file(file_name)

opts = {
    'hooks': {
       'p': [check_for_quote, check_for_header]
    }
}

six.print_(serialize.serialize(dfile.document, opts))
```

CHAPTER 2

API

Python-OOXML API

`ooxml` Package

`doc` Package

`docxfile` Package

`importer` Package

`parse` Package

`serialize` Package

CHAPTER 3

Indices and tables

- genindex
- modindex
- search