python-docx-template Documentation

Release 0.1.5

Eric Lapouyade

April 09, 2016

Contents

Quickstart

To install:

```
pip install docxtpl
```

Usage:

```python
from docxtpl import DocxTemplate

doc = DocxTemplate("my_word_template.docx")
context = { 'company_name' : "World company" }
doc.render(context)
doc.save("generated_doc.docx")
```

CHAPTER 1

Introduction

This package uses 2 major packages :

- python-docx for reading, writing and creating sub documents
- jinja2 for managing tags inserted into the template docx

python-docx-template has been created because python-docx is powerful for creating documents but not for modifying them.

The idea is to begin to create an example of the document you want to generate with microsoft word, it can be as complex as you want : pictures, index tables, footer, header, variables, anything you can do with word. Then, as you are still editing the document with microsoft word, you insert jinja2-like tags directly in the document. You save the document as a .docx file (xml format) : it will be your .docx template file.

Now you can use python-docx-template to generate as many word documents you want from this .docx template and context variables you will associate.

Note : python-docx-template as been tested with MS Word 97, it may not work with other version.

CHAPTER 2

Jinja2-like syntax

As the Jinja2 package is used, one can use all jinja2 tags and filters inside the word document. Nevertheless there are some restrictions and extensions to make it work inside a word document:

2.1 Restrictions

The usual jinja2 tags, are only to be used inside a same run of a same paragraph, it can not be used across several paragraphs, table rows, runs.

Note : a 'run' for microsoft word is a sequence of characters with the same style. For example, if you create a paragraph with all characters the same style : word will create internally only one 'run' in the paragraph. Now, if you put in bold a text in the middle of this paragraph, word will transform the previous 'run' into 3 'runs' (normal - bold - normal).

2.2 Extensions

2.2.1 Tags

In order to manage paragraphs, table rows, runs, special syntax has to be used

```
{%p jinja2_tag %} for paragraphs
{%tr jinja2_tag %} for table rows
{%r jinja2_tag %} for runs
```

By using these tags, python-docx-template will take care to put the real jinja2 tags at the right place into the document's xml source code. In addition, these tags also tells python-docx-template to remove the paragraph, table row or run where are located the begin and ending tags and only takes care about what is in between.

2.2.2 Display variables

As part of jinja2, one can used double braces:

```
{{ <var> }}
```

But if `<var>` is an RichText object, you must specify that you are changing the actual 'run'

```
{{r <var> }}
```

Note the `r` right after the openning braces

2.2.3 Cell color

There is a special case when you want to change the background color of a table cell, you must put the following tag at the very beginning of the cell

```
{% cellbg <var> %}
```

<*var*> must contain the color's hexadecimal code *without* the hash sign

2.2.4 Escaping

In order to display `{%`, `%}`, `{{` or `}}`, one can use

```
{_%, %_}, {_{ or  }_}
```

CHAPTER 3

Sub-documents

A template variable can contain a complex and built from scratch with python-docx word document. To do so, get first a sub-document object from template object and use it as a python-docx document object, see example in *tests/subdoc.py*.

CHAPTER 4

RichText

When you use `{{ <var> }}` tag in your template, it will be replaced by the string contained within *var* variable. BUT it will keep the current style. If you want to add dynamically changeable style, you have to use both : the `{{r <var> }}` tag AND a `RichText` object within *var* variable. You can change color, bold, italic, size and so on, but the best way is to use Microsoft Word to define your own *caracter* style (Home tab -> modify style -> manage style button -> New style, select 'Character style' in the form), see example in *tests/richtext.py* Instead of using `RichText()`, one can use its shortcut : `T()`

CHAPTER 5

Escaping, newline, new paragraph

When you use a `{{ <var> }}`, you are modifying an **XML** word document, this means you cannot use all chars, especially <, > and `&`. In order to use them, you must escape them. There are 3 ways :

- `context = { 'var':R('my text') }` and `{{r <var> }}` in the template (note the `r`),
- `context = { 'var':'my text'}` and `{{ <var>|e }}` in your word template
- `context = { 'var':escape('my text')}` and `{{ <var> }}` in the template.

The RichText() or R() offers newline and new paragraph feature : just use `\n` or `\a` in the text, they will be converted accordingly.

See tests/escape.py example for more informations.

CHAPTER 6

Jinja custom filters

`render()` accepts `jinja_env` optionnal argument : you may pass a jinja environment object. By this way you will be able to add some custom jinja filters:

```
from docxtpl import DocxTemplate
import jinja2

doc = DocxTemplate("my_word_template.docx")
context = { 'company_name' : "World company" }
jinja_env = jinja2.Environment()
jinja_env.filters['myfilter'] = myfilterfunc
doc.render(context,jinja_env)
doc.save("generated_doc.docx")
```

CHAPTER 7

Examples

The best way to see how it works is to read examples, they are located in *tests/* directory. Templates and generated .docx files are in *tests/test_files/*.

Functions index

Functions documentation

CHAPTER 8

Indices and tables

- genindex
- modindex
- search