docxtemplater Documentation

Release

Edgar Hipp

January 25, 2017

Contents

Contents:

Goals

Docxtemplater was born out of the idea that you should be able to generate Docx as easily as you generate Html with something like Mustache.

They are a lots of solutions like docx.js, docx4j, ... that generate docx, but you will have to write specific code to create a title, an image, ...

I think this is a waste when you can just write your template with plain old Microsoft Word.

Docxtemplater is just there for that

1.1 Why you should use a library for this

Docx is a zipped format that contains some xml. If you want to build a simple replace {tag} by value system, it can already become complicated, because the {tag} is internally separated into *<w:t>{</w:t><w:t>tag</w:t><w:t>}</w:t>*. If you want to embed loops to iterate over an array, it becomes a real hassle.

Platform Support

docxtemplater works with

- Node.js with 0.10 and 0.11

- Chrome **tested** on version 26

- Firefox 3+ (**tested** on version 21, but should work with 3+)

- Safari **tested**

- IE9+ **tested**

- Android 4.2+ **tested**

- iPads and iPhones v8.1 **tested**

You can test if everything works fine on your browser by using the test runner: http://javascript-ninja.fr/docxtemplater/v1/test/SpecRunner.html

Dependencies

1. **docxgen.js** uses [jszip.js](http://stuk.github.io/jszip/) to zip and unzip the docx files

2. Optionally, if you want to be able to name the output files, you can use **Downloadify.js**, which is required to use method download. Be informed that it uses flash, this is why the method is not recommended. This method is howewer useful because a lot of browsers are limited for the download size with the Data-URI method. **Update**: I will probably implement in the future a way to use the FileSaver API, with [FileSaverJS](http://eligrey.com/demos/FileSaver.js/)

Installation

4.1 Node

To install docxtemplater, we recommend you to use npm.

```
npm install docxtemplater
```

If you want to use the command line interface, you should use the global flag, eg:

```
npm install docxtemplater -g
```

4.2 Browser

I recommend you to use browserify.

```
git clone git@github.com:edi9999/docxtemplater.git && cd docxtemplater
npm install -g gulp jasmine-node uglify-js
npm install
gulp allCoffee
mkdir build -p
browserify -r ./js/docxgen.js -s Docxgen > build/docxgen.js
uglifyjs build/docxgen.js > build/docxgen.min.js
```

The -s Docxgen will export docxgen to window.Docxgen for easy usage

Your version of docxtemplater will be in /build (minified and non minified options).

Syntax

The syntax is highly inspired by Mustache. The template is created in Microsoft Word or any equivalent that saves to docx.

5.1 Synopsis

A typical docxtemplater template:

```
Hello {name} !
```

Given the following hash:

```
{
    name:'Edgar'
}
```

Will produce:

```
Hello Edgar !
```

5.2 Tag types

Like Mustache, it has the loopopening {#} and loopclosing {/} brackets

5.3 Loop syntax

The following template:

```
{#products}
    {name}, {price} €
{/products}
```

Given the following hash:

```
{
    "products":
        [
        {name:"Windows",price:100},
```

```
        {name:"Mac OSX",price:200},
        {name:"Ubuntu",price:0}
      ]
}
```

will result in :

```
Windows, 100 €
Mac OSX, 200 €
Ubuntu, 0€
```

The loop behaves in the following way:

- If the value is an array, it will loop over all the elements of that array.

- If the value is a boolean, it will loop once if the value is true, keeping the same scope, and not loop at all if the value is false

---

Note: Because the loops work also with boolean values, you can also use them for conditions.

5.4 Dash syntax

It is quite difficult to know on which element you are going to loop. By default, when using the for loop, docxgen will find that by himself:

If between the two tags {#tag}_____{/tag}

- they is the Xml Tag <w:tc> -> you are in a table, and it will loop over <w:tr>

- else -> it will loop over <w:t>, which is the default Text Tag

With the Dash syntax you pass as a first argument the tag you want to loop on:

```
{-w:p loop} {inner} {/loop}
```

In this case this will loop over the first parent <w:p> tag

5.5 Inverted Selections

An inverted section begins with a caret (hat) and ends with a slash. That is {^person} begins a "person" inverted section while {/person} ends it.

While sections can be used to render text one or more times based on the value of the key, inverted sections may render text once based on the inverse value of the key. That is, they will be rendered if the key doesn't exist, is false, or is an empty list.

Template:

```
{#repo}
  <b>{name}</b>
{/repo}
{^repo}
  No repos :(
{/repo}
```

Hash:

```
{
  "repo": []
}
```

Output:

```
No repos :(
```

Generate a document

Here's a sample code to generate a document:

```
//Only for Node Usage
DocxGen=require('docxtemplater');
content=fs.readFileSync(__dirname+"/input.docx","binary")

doc=new DocxGen(content);
doc.setData({
    "first_name":"Hipp",
    "last_name":"Edgar",
    "phone":"0652455478",
    "description":"New Website"
}) //set the templateVariables
doc.render() //apply them (replace all occurences of {first_name} by Hipp, ...)
zip=doc.getZip() //Get the zip representation of the docx

//Only for Node Usage
output=doc.getZip().generate({type:"nodebuffer"})
fs.writeFileSync("output.docx",output)
```

Configuration

Here are documented the special options that you can set when creating a new DocxGen to get some more superpower :

It documents the options parameter when you do:

```
var doc=new DocxGen(content);
doc.setOptions(options)
```

7.1 Image Replacing

Note: The imageReplacing feature has been removed from the main docxtemplater package. This feature will be implemented in the future in an external module.

7.2 Custom Parser

The name of this option is *parser* (function).

With a custom parser you can parse the tags to for example add operators like '+', '-', or whatever the way you want to parse expressions. See for a complete reference of all possibilities of angularjs parsing: http://teropa.info/blog/2014/03/23/angularjs-expressions-cheatsheet.html

To enable this, you need to specify a custom parser. You need to create a parser function:

docxtemplater comes shipped with this parser:

```
parser=function(expression)
{
    return {
        get:function(scope) {
            if (expression===".") return scope;
            return scope[expression]
        }
    };
}
```

To use the angular-parser, do the following:

```
expressions= require('angular-expressions');
// define your filter functions here, eg:
// expressions.filters.split = function(input, str) { return input.split(str); }
angularParser= function(tag) {
    return {
        get: tag == '.' ? function(s){ return s;} : expressions.compile(tag)
    };
}
new DocxGen(data).setOptions({parser:angularParser})
```

Note: The require() works in the browser if you include vendor/angular-parser-browser.js

7.3 Intelligent LoopTagging

The name of this option is *intelligentTagging* (boolean).

When looping over an element, docxtemplater needs to know over which element you want to loop. By default, it tries to do that intelligently (by looking what XML Tags are between the {tags}). However, if you want to always use the <w:t> tag by default, set this option to false.

You can always specify over which element you want to loop with the dash loop syntax

Command Line Interface (CLI)

This section is about the commandline interface of docxtemplater.

The syntax is the following:

docxtemplater config.json

The full config.json should be something like that:

```
{
    "config.docxFile":"input.docx",
    "config.outputFile":"output.docx",
    "config.qrcode":true,
    "config.debug":true,
    "first_name":"John",
    "last_name":"Smith",
    "age":62
}
```

8.1 Config.json Syntax

8.1.1 Config properties:

These are the properties to configure docxtemplater:

```
{
    "config.docxFile":"input.docx", //The input file path
    "config.outputFile":"output.docx", //The output file path
    "config.qrcode":true, //whether the images should be scanned to replace them by qrcodes (slows do
    "config.debug":true //whether to show debug output or not
}
```

8.1.2 Data properties:

To add data to your template, just use keys that don't start with "config."

```
{
    "first_name":"John",
    "last_name":"Smith",
    "age":62
}
```

Full Documentation per method

9.1 Creating a new Docxgen Object

```
new DocxGen()

    This function returns a new DocxGen Object

new DocxGen(content,options)

    if content is defined, it will call `.load(content,options)`
```

9.2 Docxgen methods

```
load(content,options)

    This will call new JSzip().load(content,options) under the hood. See http://stuk.github.io/jszip,
    You can also pass a JSzip object as the first argument.

setData(Tags)

    Tags:
        Type: Object {tag_name:tag_replacement}
        Object containing for each tag_name, the replacement for this tag. For example, if you want t

render()

    This function replaces all template variables by their values

getZip()

    This will return you the zip that represents the docx. You can then call `.generate` on this to g

getFullText:([path])

    path
        Type:"String"
        Default:"word/document.xml"
        This argument determines from which document you want to get the text. The main document is c

    @returns
```

```
        Type:"String"
        The string containing all the text from the document

    This method gets only the text of a given document (not the formatting)

getTags()

    This function returns the template variables contained in the opened document. For example if the

        {name}
        {first_name}
        {phone}

    The function will return:
        [{
            filename:"document.xml",
            vars:
            {
                name:true,
                first_name:true,
                phone:true
            }
        }]

    If the content contains tagLoops:

        {title}
        {#customer}
        {name}
        {phone}
        {/customer}


    The function will return:

        [{
            filename:"document.xml",
            vars:
            {
                title:true,
                customer:
                {
                    name:true,
                    phone:true
                }
            }
        }]
```

Copyright

Demos

Including:

Indices and tables

- genindex
- modindex
- search

C

Command Line Interface (CLI), 18
Configuration, 15
Copyright, 22

F

Full_doc, 19

G

Generate a Document, 13
Goals, 1

I

Installation, 7

P

platform_support, 3

S

Syntax, 9