docxtemplater Documentation

Release

Edgar Hipp

January 25, 2017

Contents

Contents:

CHAPTER 1

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.

CHAPTER 2

Platform Support

docxtemplater works with

- Node.js versions 0.10, 0.11, 0.12 ,4, 5 and 6, with iojs
- 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/v3/test/SpecRunner.html

CHAPTER 3

Dependencies

1. [jszip.js](http://stuk.github.io/jszip/) to zip and unzip the docx files
2. [xmldom](https://github.com/jindw/xmldom) to parse the files as xml

CHAPTER 4

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 the npm scripts I wrote (which can be found in the package.json).

```
git clone git@github.com:edi9999/docxtemplater.git && cd docxtemplater
npm install
npm run compile
# Optionally :
# npm run browserify
# npm run uglify:lib
```

Docxtemplater will be exported to window.Docxtemplater for easy usage (on some systems, it might export it in window.docxtemplater (see https://github.com/edi9999/docxtemplater/issues/118))

Your version of docxtemplater will be in /build (minified and non minified options) and already include all dependencies

CHAPTER 5

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 data:

```
{
    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 data:

```
{
    "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

5.4 Conditionals

Because the loops work with boolean values, you can also use them for conditions. This allows optional fields in your data. For example, this template:

```
{#people}
First name: {first_name}
Last name: {last_name}
{#haskitty}Cat's name: {kitty}{/haskitty}{/people}
```

and this data:

yield the following:

```
First name: Jane
Last name: Doe

First name: John
Last name: Roe
Cat's name: Chairman Meow
```

5.5 Dash syntax

It is quite difficult to know on which element you are going to loop. By default, when using the for loop, docxtemplater 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

5.6 Raw Xml syntax

Sometimes, you would like to insert your custom XML (a complex table, a formula, ...)

With the RawXml syntax the variable is interpreted as XML and replaced in the formula

```
{@rawXml}
```

with this data:

```
{rawXml:'<w:p><w:pPr><w:rPr><w:color w:val="FF0000"/></w:rPr></w:pPr><w:r><w:rPr><w:color w:val="FF0
```

This will loop over the first parent <w:p> tag

5.7 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}
```

Data:

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

Output:

```
No repos :(
```

CHAPTER 6

Generate a document

Here's a sample code to generate a document:

```
var fs=require('fs')
var JSZip = require('jszip');
var Docxtemplater = require('docxtemplater');

//Load the docx file as a binary
var content = fs
    .readFileSync(__dirname+"/input.docx","binary")

var zip = new JSZip(content);

var doc=new Docxtemplater();
doc.loadZip(zip);

//set the templateVariables
doc.setData({
    "first_name":"Hipp",
    "last_name":"Edgar",
    "phone":"0652455478",
    "description":"New Website"
});

//apply them (replace all occurences of {first_name} by Hipp, ...)
doc.render();

var buf = doc.getZip()
             .generate({type:"nodebuffer"});

fs.writeFileSync(__dirname+"/output.docx",buf);
```

CHAPTER 7

Configuration

The options that you can set when creating a new Docxtemplater

It documents the options parameter when you do:

```
var doc=new Docxtemplater();
doc.loadZip(zip);
doc.setOptions(options)
```

7.1 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.

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(tag) {
  return {
    'get': function(scope) {
      if (tag === '.') {
        return scope;
      }
      else {
        return scope[tag];
      }
    }
  };
},
```

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 Docxtemplater().loadZip(zip).setOptions({parser:angularParser})
```

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

See for a complete reference of all possibilities of angularjs parsing: http://teropa.info/blog/2014/03/23/angularjs-expressions-cheatsheet.html

7.2 Custom delimiters

You can set up your custom delimiters with this syntax:

```
new Docxtemplater()
    .loadZip(zip)
    .setOptions({delimiters:{start:'[[',end:']]'}})
```

7.3 nullGetter

You can customize the value that is shown whenever the parser returns 'null' or undefined. The nullGetter option is a function By default the nullGetter is the following function

```
nullGetter(part) {
        if (!part.module) {
                return "undefined";
        }
        if (part.module === "rawxml") {
                return "";
        }
        return "";
},
```

This means that the default value for simple tags is to show "undefined". The default for rawTags ({@rawTag}) is to drop the paragraph completely (you could enter any xml here).

7.4 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

7.5 Image Replacing

Note: The imageReplacing feature has been removed from the main docxtemplater package. This feature has been implemented in an external module that can be found here : https://github.com/open-xml-templating/docxtemplater-image-module.

CHAPTER 8

Possible Errors

This section is about the possible errors that Docxtemplater will throw

8.1 Schema of the error

All errors thrown by docxtemplater have the following schema:

8.2 Error example

If your content is *{user {name}*, docxtemplater will throw the following error :

try doc.render()

catch e e.name==”TemplateError” e.message==”Unclosed tag” e.properties.explanation==”The tag beginning with ‘{user ‘ is unclosed” e.properties.id==”unclosed_tag” e.properties.context==”{user {” e.properties.xtag==”user “

CHAPTER 9

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 the following:

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

9.1 Config.json Syntax

9.1.1 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.debug":true //whether to show debug output or not
}
```

9.1.2 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
}
```

CHAPTER 10

Full Documentation per method

10.1 Creating a new Docxtemplater Object

```
new Docxtemplater()

    This function returns a new Docxtemplater Object
```

10.2 Docxtemplater methods

```
loadZip(zip)

    You have to pass a zip instance to that method, coming from jszip.

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
```

CHAPTER 11

Frequently asked questions

11.1 How to make docxtemplater work with IE9

docxtemplater should work on almost all browsers as of version 1 : IE7 + . Safari, Chrome, Opera, Firefox.

The only 'problem' is to load the binary file into the browser. This is not in docxtemplater's scope, but here is the code that jszip's creator recommends to use to load the zip from the browser:

https://stuk.github.io/jszip/documentation/howto/read_zip.html

The following code should load the binary content on all browsers:

```
JSZipUtils.getBinaryContent('path/to/content.zip', function(err, data) {
  if(err) {
    throw err; // or handle err
  }

  var zip = new JSZip(data);
});
```

11.2 How to insert linebreaks

```
pre = '<w:p><w:r><w:t>';
post = '</w:t></w:r></w:p>';
lineBreak = '<w:br/>';
text = pre + 'testing line 1' + lineBreak + 'testing line 2' + post;
data = {text : text}
docx.setData(data)
```

then in your template, just put {@text} instead of the usual {text}

If you use the angular-parser, you can also write a filter like this:

```
angularexpressions.filters.raw = function (text) {
    var lines = text.split("\n");
    var pre = "<w:p><w:r><w:t>";
    var post = "</w:t></w:r></w:p>";
    var lineBreak = "<w:br/>";
    return pre + lines.join(lineBreak) + post;
}
```

and then have your docx as : {@text|raw}

with

```
data = {text: "testing line 1 \n testing line 2"};
docx.setData(data)
```

11.3 Smaller docx output using compression

The size of the docx output can be big, in the case where you generate the zip the following way:

```
docx.getZip().generate({ type: "nodebuffer"})
```

This is because the zip will not be compressed in that case. To force the compression (which could be slow because it is running in JS for files bigger than 10 MB)

```
var zip = docx.getZip().generate({
        type: "nodebuffer",
        compression: "DEFLATE"
});
```

CHAPTER 12

Demos

Including:

- Replace Variables
- Formating
- Angular Parsing
- Loops
- Loops and tables
- Lists
- Replacing images
- Raw Xml Insertion

CHAPTER 13

Indices and tables

- genindex
- modindex
- search