# docxtemplater Documentation

***Release 0.5.4***

**Edgar Hipp**

June 19, 2014

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

# Installation

## 2.1 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
```

## 2.2 Browser

You will just have to include the docxgen.js or docxgen.min.js file.

CHAPTER 3

# Syntax

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

## 3.1 Synopsis

A typical docxtemplater template:

```
Hello {name} !
```

Given the following hash:

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

Will produce:

```
Hello Edgar !
```

## 3.2 Tag types

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

## 3.3 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

## 3.4 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

## 3.5 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 :(
```

CHAPTER 4

# Generate a document

Here's a sample code to generate a document:

```
DocxGen=require('docxtemplater'); //Only for Node Usage
new DocxGen().loadFromFile("tagExample.docx",{async:true}).success(function(doc){
        doc.setTags({
        "first_name":"Hipp",
                "last_name":"Edgar",
                "phone":"0652455478",
                "description":"New Website"
            }) //set the templateVariables
        doc.applyTags() //apply them (replace all occurences of {first_name} by Hipp, ...)
        doc.output() //Output the document using Data-URI
});
```

The different steps are the following:

## 4.1 Loading a document:

A docx can be loaded by it's base64 representation, like this:

```
doc=new DocxGen(base64Data,options);
```

However, loading the base64Data is not that easy, so I created a wrapper function to load a docx from a file via Ajax.

**Note:** The options parameter will be explained later on.

```
doc=new DocxGen().loadFromFile(documentPath,options);
```

The documentPath is the path to the document you want to load. The options are the same as in the constructor function, and I've added the async parameter.

- If async is true, the document is loaded asynchronously and returns a promise (eg you can write .success(callback) to get when the document is loaded)
- If async is false, the document is loaded synchronously and returns the document.

## 4.2 Setting the tags:

The tags are the variables that are going to be replaced by their values. To set them, just use:

```
doc.setTags(tags);
```

## 4.3 Applying the tags:

Applying the tags means opening all files that contain text (eg: footer, header, main document), and replace the variables by their values.

```
doc.applyTags(tags)
```

---

**Note:** If you specify an argument for the applyTags method, the function setTags(tags) will be called before applying the tags.

---

## 4.4 Outputing the document:

They are several ways to output the document. The most basic usage is to download the document.

```
doc.output(options)
```

Depending on your environment, if you don't set any options, this will:

- In the browser: Download the document using DataURI
- In Node: Save the document with the given fileName (output.docx by default)

Here's the different options parameters:

```
name:
    Type:string["output.docx"]
    The name of the file that will be outputed (doesnt work in the browser because of dataUri downloa

callback:
   Type:function
   Function that is called without arguments when the output is done. Is used only in Node (because i

download:
    Type:boolean[true]
    If download is true, file will be downloaded automatically with data URI.
    returns the output file.

type:
    Type:string["base64"]
    The type of zip to return. The possible values are : (same as in http://stuk.github.io/jszip/ @ge
    base64 (default) : the result will be a string, the binary in a base64 form.
    string : the result will be a string in "binary" form, 1 byte per char.
    uint8array : the result will be a Uint8Array containing the zip. This requires a compatible brows
    arraybuffer : the result will be a ArrayBuffer containing the zip. This requires a compatible bro
    blob : the result will be a Blob containing the zip. This requires a compatible browser.
    nodebuffer : the result will be a nodejs Buffer containing the zip. This requires nodejs.
```

This function creates the docx file and downloads it on the user's computer. The name of the file is download.docx for Chrome, and some akward file names for Firefox: VEeTHCfS.docx.part.docx, and can't be changed because it is handled by the browser. For more informations about how to solve this problem, see the **Filename Problems** section on [http://stuk.github.io/jszip/](http://stuk.github.io/jszip/)

---

**Note:** Note: All browsers don't support the download of big files with Data URI, so you **should** use the *download* method for files bigger than 100kB

CHAPTER 5

# Special options

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

## 5.1 Image Replacing

To stay a templating engine, I wanted that DocxTemplater doesn't add an image from scratch, but rather uses an existing image that can be detected, and DocxTemplater will just change the contents of that image, without changing it's style. The size of the replaced images will stay the same, ...

So I decided to use the qrcode format, which is a format that lets you identify images by their content.

**Note:** If you don't use that functionality, you should disable it, because it is quite slow (the image decoding)

## 5.2 Angular Parser

You can set the angular parser with the following code:

CHAPTER 6

# Configuration

CHAPTER 7

# 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
}
```

## 7.1 Config.json Syntax

### 7.1.1 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
}
```

### 7.1.2 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 8

# Copyright

License

(The MIT license)

CHAPTER 9

# Demos

Including:

- Replace Variables
- Formating
- Angular Parsing
- Loops
- Loops and tables
- Lists
- Replacing images
- Naming the output
- Using QrCodes
- Replacing many images with QrCode
- Raw Xml Insertion

CHAPTER 10

# Indices and tables

- *genindex*
- *modindex*
- *search*