

---

# **docxtemplater Documentation**

***Release***

**Edgar Hipp**

January 25, 2017



<b>1</b>	<b>Goals</b>	<b>3</b>
1.1	Why you should use a library for this . . . . .	3
<b>2</b>	<b>Platform Support</b>	<b>5</b>
<b>3</b>	<b>Dependencies</b>	<b>7</b>
<b>4</b>	<b>Installation</b>	<b>9</b>
4.1	Node . . . . .	9
4.2	Browser . . . . .	9
<b>5</b>	<b>Syntax</b>	<b>11</b>
5.1	Synopsis . . . . .	11
5.2	Tag types . . . . .	11
5.3	Loop syntax . . . . .	11
5.4	Dash syntax . . . . .	12
5.5	Inverted Selections . . . . .	12
<b>6</b>	<b>Generate a document</b>	<b>15</b>
<b>7</b>	<b>Configuration</b>	<b>17</b>
7.1	Image Replacing . . . . .	17
7.2	Custom Parser . . . . .	17
7.3	Custom delimiters . . . . .	18
7.4	Intelligent LoopTagging . . . . .	18
<b>8</b>	<b>Command Line Interface (CLI)</b>	<b>19</b>
8.1	Config.json Syntax . . . . .	19
<b>9</b>	<b>Full Documentation per method</b>	<b>21</b>
9.1	Creating a new Docxgen Object . . . . .	21
9.2	Docxgen methods . . . . .	21
<b>10</b>	<b>Copyright</b>	<b>23</b>
<b>11</b>	<b>Demos</b>	<b>25</b>
<b>12</b>	<b>Indices and tables</b>	<b>27</b>



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.

There are a lot 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



---

## 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 (on some systems, it might export it in window.docxgen (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



---

## 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:

```
fs=require(`fs`)
Docxtemplater = require('docxtemplater');

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

doc=new Docxtemplater(content);

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

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

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

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



---

## 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 has been implemented in an external module that can be found here : <https://github.com/edi9999/docxtemplater-image-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 Custom delimiters

You can set up your custom delimiters with this syntax:

```
new DocxGen(data)
  .setOptions({delimiters:{start:'[[',end:']]'}})
```

**Warning:** In previous versions, you could write `DocUtils.Tags={start:'[[',end:']]'};`, but this is deprecated now and will be removed in future versions.

## 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

---

## 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.inputFile": "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 d
  "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 c

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**

---

License

(The MIT license)

Copyright (c) 2013 Edgar Hipp

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



### 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



---

**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