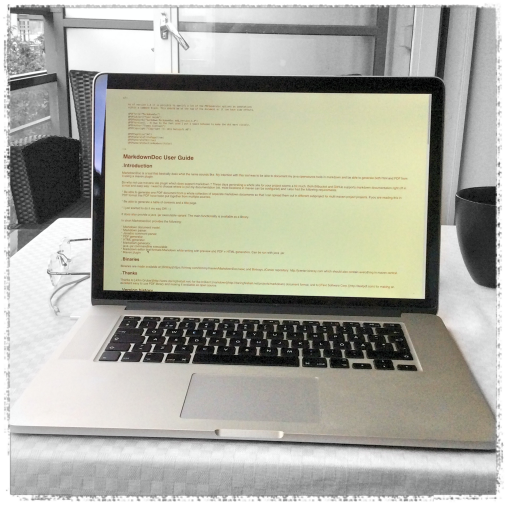
# MarkdownDoc

## User Guide

2.0.1



Tommy Svensson

# MarkdownDoc User Guide

## *Introduction*

MarkdownDoc is a tool that basically does what the name sounds like. My intention with this tool was to be able to document my java opensource tools in markdown and be able to generate both html and PDF from it using a maven plugin.

So why not use mavens site plugin which does support markdown ? These days generating a whole site for your project seems a bit much. Both Bitbucket and GitHub supports markdown documentation right off in a nice and easy way. I want to choose where to put my documentation (ok, most locations in maven can be configured) and I also had the following requirements:

- Be able to generate one PDF document from a whole collection of separate markdown documents so that I can spread them out in different subproject for multi maven project projects. If you are reading this in PDF format this PDF have been put together from multiple sources.
- Be able to generate a table of contents and a title page.
- I just wanted to do it my way OK! :-)

It does also provide a java -jar executable variant. The main functionality is available as a library.

In short MarkdownDoc provides the following:

- Markdown document model.
- Markdown parser.
- Javadoc comment parser.
- PDF generator.
- HTML generator.
- Markdown generator.
- java -jar commandline executable.
- Markdown editor that formats Markdown while writing with preview and PDF + HTML generation. Can be run with java -jar.
- Maven plugin.

## Binaries

Binaries are made available at Bintray and Bintrays JCenter repository: http://jcenter.bintray.com which also mirrors everything in maven central.

## Thanks

Thanks to John Gruber for the brilliant markdown document format, and to PDFBox Apache Project for making a completely open source PDF renderer under the sensible Apache 2.0 license.

## How markdown is MarkdownDoc ?

Well, it implements the "specification" as documented on daringfireball.net. This specification however is not extremely exact so there might be some differences.

The known (and intentional) differences are:

- No HMTL pass-through! Well, there is a small exception to that. HTML comments are passed along. Mostly because there is no markdown comment format and I wanted to be able to put comments in my documents. " " is passed through to create indents that are not code blocks. `"<div class="...">...</div>"` is also passed through. The reason for no general HTML pass-through is that MarkdownDoc takes it directly from markdown to a document model which is then used to generate PDF without any HTML rendering in between. The main purpose of this tool is to write documentation not generate HTML sites (though that has become easier in version 1.4 with the div support).

- Escaping with '\'. In MarkdownDoc you can escape any character with \ and it will be passed through as is without being acted on if it has markdown meaning.
- No entity encoding of email addresses.
- No multiple block quote levels (as of now). I've never personally missed having multiple quote levels, which is why I haven't done something about that yet. No one has contacted me asking for it either :-).
- Does not support any other formatting within strong, emphasized, or header. I personally don't see enough of a problem with this, that I'll prioritize it.

If you find any of the missing features a problem, I'll happily accept pull requests. :-) Seriously, it is OK to contact me with functionality wishes. Do note however that I work on this and other projects entirely in my spare time.

## File specifications

With both the maven plugin and the command line execution jar file you can specify a set of files to use as input. These are basically a comma separated list of files, but with the following additions:

/my/path

*All files in the directory pointed to by the path.*

/my/path/**

*All files in the directory pointed to by the path and sub directories.*

/my/path/**/*regexp pattern*

*All files matching the pattern in the directory pointed to by the path and sub directories.*

/my/path/*regexp pattern*

*All files matching the pattern in the directory pointed to by the path.*

/my/path/fileset.fs

*The above rules are applied to all file specifications in files having the .fs extension. # are comment lines within .fs files.*

# Command Line

## *General*

MarkdownDoc can be run using `java -jar markdowndoc-cmd-line-n.n[.n]-exec.jar`. If you just run it without any arguments you get the following:

```
Usage: java -jar markdowndoc-cmd-line-n.n[.n].exec.jar <generator> --help
       or
       java -jar markdowndoc-cmd-line-n.n[.n].exec.jar <generator> <fileSpec>
           --<generator option> ...
       or
       java -jar markdowndoc-cmd-line-n.n[.n].exec.jar <generator> <fileSpec>
           parserOptions:<parserOptions> —-<generator option> ...
       or
       java -jar markdowndoc-cmd-line-n.n[.n].exec.jar <path to a .mddoc file>
```

The last usage example requires an *.mddoc* file. See *'The* mddoc *file type'* (section 5) for more information on this file type.

What the generator options are depends on the specified generator.

The markdowndoc-cmd-line-n.n[.n]-exec.jar is a jar generated to contain all dependencies in the same jar, making it easy to execute with java -jar.

The <*generator*> part should be either *pdf*, *html*, or *md*.

The <*filespec/*> part is a comma separated list of paths relative to the current directory. The filename part of the path can contain regular expressions and the directory part of the path can specify `.../**/...` to mean any levels of subdirectories.

Example: `root/**/docs/.*.md`

See "Settings / Options" elsewhere in this document for all the options to the different generators and parsers.

# Maven Plugin

The maven plugin is rather straight forward. It has 3 sets of configuration structures, one common and one for each generator.

## *generatorOptions*

There is a config section that is common to all generators and specifys which generator to run and what input files to include. The following example is from the generation of this manual:

```
    <generatorOptions>
        <generator>pdf</generator>
        <inputPaths>
            Docs/parts/H1UserGuide.mdpart,
            Docs/MarkdownDoc.md,
            MavenPlugin/docs/.*.md,
            CommandLine/docs/.*.md,
            Library/docs/.*.md,
            Docs/parts/H1Licenses.mdpart,
            Docs/licenses.md,
            Docs/parts/H1LicenseTexts.mdpart,
            Docs/.*-.*.md
        </inputPaths>
        <parserOptions>option=value,....</parserOptions>
    </generatorOptions>
```

If the `<inputPaths>...</inputPaths>` section only contain one file of type *.mddoc* then no other parameters need to be specified, not even `<generator>...</generator>`! In this case all information needed to generate final documents resides in the *.mddoc* file. See *'The* mddoc *file type'* elsewhere in this document for more information on this file type.

The current valid argument for `<generator>...</generator>` are *pdf*, *html*, and *md*.

The input paths are comma separated and are always relative to the root of the maven project. To clarify that, for a multi module maven build it is always the top root with the top pom that is the root even if you start the build at a lower level. This root is resolved by starting at *${basedir}* and going up until the parent directory does not have a pom. I have found no way to let maven tell me this path.

The paths can have wildcards in form of regular expressions for the file names. There is also a special directory name ** that means any level of subdirectories.

All the input paths are parsed into the same document model that then gets passed to the generator. They are parsed in the order they are specified. When it comes to wildcards it is hard to say which order they will be in. It might differ on different platforms.

## *Example*

Following is a complete plugin specification with all options specified:

```
    <plugin>
        <groupId>se.natusoft.tools.doc.markdowndoc</groupId>
        <artifactId>markdowndoc-maven-plugin</artifactId>
        <version>n.n[.n]</version>

        <executions>
            <execution>
                <id>generate-docs</id>
                <goals>
                    <goal>doc</goal>
                </goals>
                <phase>install</phase>
                <configuration>

                    <generatorOptions>
                        <generator>pdf|html|md</generator>
```

```
                        <inputPaths>
                            ...
                        </inputPaths>
                    </generatorOptions>

                    <pdf|html|mdGeneratorOptions>
                        ...
                    </pdf|html|mdGeneratorOptions>

                </configuration>
            </execution>
        </executions>
    </plugin>
```

See the "Options / Settings" part elsewhere in this document for all the specific options.

# HTML Support

As said in the introduction, MarkdownDoc generates both HTML and PDF, but in either case it goes directly from parsing the markdown to generating the target format using a markdown document model between parser and generators. That means that HTML is just one target format. It is not an intermediate between other target formats. Thereby only markdown, and not HTML is supported in input. There are however a few exceptions to that.

##  

This is recognized as a special space. It will not be reacted on for code blocks for example. Thereby it can be used to indent text without creating a code block when the indent reaches 4 spaces.

## HTML comments

These are recognized, parsed and passed along in model. It is upp to generator to decide to generate comments or not. The HTMLGenerator do generate comments, as do the MarkdownGenerator. The PDFGenerator does not for obvious reasons.

I think it is nice to be able to have comments in documents. Comment blocks are also used to hide other MarkdownDoc special features also. For example options annotations.

## Divs

HTML div tags are supported and will be generated by HTMLGenerator. But the PDFGenerator can also make use of div tags. As of version 1.4 the PDFGenerator makes use of a JSON based markdown stylesheet I've called MSS. In MSS you can also define styles in divs that will only apply to text within those div blocks. This allows for great styling flexibility also to PDF documents.

Due to a GitHub feature that stops markdown rendering between divs in markdown documents I've implemented 2 ways of specifying the divs:

```
<div class="class">
    ...
</div>
```

or

```
<!-- @Div("class") -->
    ...
<!-- @EndDiv -->
```

The latter does not affect GitHub. In either usage the HTMLGenerator will generate a correct HTML div, while the MarkdownGenerator will generate the comment variant, and the PDFGenerator will not generate anything, but will use the divs to affect styling via MSS.

The div has to be the only thing on the line, both start and end tag! Also, there can be a maximum of 3 spaces before the div tag. If there are 4 or more it will become a code block!

**Note:** that the HTML div variant will have to look exactly like the example above. Only `class="..."` is allowed in the tag! The start div will not be recognized if you add more to it, and there will be confusion when the end tag is seen in that case.

# Library

The library is made up of a document model representing all formats of markdown, parsers and generators. The parsers produce a document model and the generators generate from that model. The document model represents the markdown formats. Thereby there are no HTML pass-through from a markdown document! This tool only deals with markdown, not HTML.

The API docs for the library can be found here.

## *Usage*

In package se.natusoft.doc.markdown.api there are 3 API classes:

**Options** - This represents options for a generator. It should be seen as a narrow variant of Object representing only generator options, but any such. It has one method common to all `public boolean isHelp()`. Implementations should have a default constructor.

**Parser** - This represents a parser.

```
public interface Parser {
    public void parse(Doc document, File parseFile, Properties parserOptions) throws IOException,
    ParseException;
}
```

The parser gets passed an already created Doc model allowing the document to be built from multiple source files by parsing into the same document.

**Generator** - This represents a generator.

```
public interface Generator {
    public Class getOptionsClass();
    public void generate(Doc document, Options options, File rootDir) throws IOException,
    GenerateException;
}
```

*getOptionsClass()* returns the class implementing Options and holding all the options for the generator.

*generate(...)* generates the document provided by *document* using the specified *options* and producing the result in whatever *rootDir* relative path is specified in the *options*.

# Parsers

## se.natusoft.doc.markdown.parser.MarkdownParser

This parser parses markdown and only markdown! It ignores HTML with the exception of comments,   and divs.

Example usage:

```
Parser parser = new MarkdownParser();
Doc document = new Doc();
Properties parserOptions = new Properties();
parser.parse(document, parseFile, parserOptions);
```

## se.natusoft.doc.markdown.parser.JavadocParser

This parser parses java source files (should also handle groovy source files) and extracts class and method declarations and javadoc comment blocks. It produces a document model looking like this (in

markdown format):

```
public _class/interface_ __class-name__ extends something [package] {
> class javadoc

__full method declaration__
> method javadoc
_Returns_
> description
_Parameters_
> _param_ - description
_Throws_
> _exception_ - description
_See_
> description

...
}
```

This allows you to include API documentation in your documentation without having to duplicate it. Please note that if `markdownJavadoc=true` parser option have been specified then *class javadoc* and *method javadoc* will not be formatted but passed to the MarkdownParser instead.

Example usage:

```
Parser parser = new JavadocParser();
Doc document = new Doc();
Properties parserOptions = new Properties();
parser.parse(document, parseFile, parserOptions);
```

## se.natusoft.doc.markdown.parser.ParserProvider

This is a utility to get a parser based on file extension. ".md", ".markdown", ".mdpart", and ".java" are valid extensions that will return a parser. If the passed file does not have a valid extension null will be returned.

Example usage:

```
Parser parser = ParserProvider.getParserForFile(parseFile);
Doc document = new Doc();
Properties parserOptions = new Properties();
parserOptions.setProperty("...", "...");
parser.parse(document, parseFile, parserOptions);
```

# Generators

Example usage:

```
public static void main(String[] args) {
    Doc document = new Doc();

    ... parsing of document.

     Generator generator = new [PDFBox|HTML|Markdown]Generator();

     // I'm using OptionsManager to load the options in this example.
     // If you use maven or ant then those tools will have loaded
     // the options for you and getOptionsClass() is not relevant
     // in that case.
    CommandLineOptionsManager<Options> optMgr =
        new CommandLineOptionsManager<Options>(generator.getOptionsClass());
    Options options = optMgr.loadOptions("--", args);
    if (options.isHelp()) {
        optMgr.printHelpText("--","", System.out);
    }
    else {
```

```
            File rootDir = new File();
            generator.generate(document, options, rootDir);
        }
    }
```

Please note that the CommandLineOptionsMangager used in the example is part of the OptionsManager tool also by me. Available at github.com/tombensve/OptionsManager.

## se.natusoft.doc.markdown.generator.PDFBoxGenerator

This generator produces a PDF document.

## se.natusoft.doc.markdown.generator.HTMLGenerator

This generator produces an HTML document.

## se.natusoft.doc.markdown.generator.MarkdownGenerator

This generator produces a Markdown document. So why would we want to generate markdown ? Well, it became needed after I added the JavadocParser. Now I can have both markdown and java files as input and the PDF and HTML files contains the whole result including the javadoc information. The original markdown document however does not have the javadoc parts, and this markdown document is read as is on github and will then not be complete. Therefore I added this generator and moved my real source document into docs/src and also generate a markdown version into docs that will be as complete as the pdf and html version.

## se.natusoft.doc.markdown.util.MDDocFileHandler

This is a class with one static method that completely handles the *.mddoc* format.

Usage:

```
    MDDocFileHandler.execute("<path to .mddoc file>");
```

This will generate all output formats as specified in the .mddoc file.

See the "The mddoc file type" section for more information on the .mdddoc format.

# *Bugs*

Nothing currenty known by me.

# MSS (Markdown(Doc) Style Sheet)

The MSS format is a JSON document describing the styles (colors and fonts) to use for different sections of a markdown document (standard text, heading, bold, code, etc). It contains 3 main sections: front page, TOC, pages. There is a *default.mss* embedded in the jar that is used if no external mss files is provided. The default MSS have changed in version 2.0.0 and now produces output that looks different than previous versions. Not only different, but better IMHO :-). It still defaults to A4 page size, but now also have correct margins according to standards. Maybe iText also did that, but it feels like the margins are larger now (2.54 cm).

Note that page size is now set in the MSS file and not provided as an option when generating. The margins are of course also set in MSS.

Currently the MSS file is only used by the PDF generator. But it could also be relevant for other formats, like word if I ever add such a generator. I gladly take a pull request if anybody wants to do that :-).

Here is an example of an MSS file with descriptions:

```
    {
```

This section is specific to PDF files, and specifies ttf, otf, and other font types supported by iText. For the *internal* fonts "HELVETICA" or "COURIER" is specified as "family", but to use a font specified here, just use the name set as "family" here. If you are using an exernal Helvetica font specified here, dont call it just "HELVETICA" since there will be confusion!

A best effort is used to resolve the font in "path":. If the specified path does not match relative to current directory then it will try the parent directory and so on all the way upp to the filesytem root.

```
      "pdf": {
        "extFonts": [
          {
            "family": "MDD-EXAMPLE",
            "encoding": "UTF-8",
            "path": "/fonts/ttf/some-font.ttf"
          }
        ]
      },
```

The "colors" section just provide names for colors. This list was taken from the default color names for CSS colors, with the exception of the first 3. Any color specification in sections below that does not contain any ":" character will be taken as a name and looked up here.

```
      "colors": {
        "white": "255:255:255",
        "black": "0:0:0",
        "mddgrey": "128:128:128",
        "AliceBlue": "F0F8FF",
        "AntiqueWhite": "FAEBD7",
        "Aqua": "00FFFF",
        "Aquamarine": "7FFFD4",
        "Azure": "F0FFFF",
        "Beige": "F5F5DC",
        "Bisque": "FFE4C4",
        "Black": "000000",
        "BlanchedAlmond": "FFEBCD",
        "Blue": "0000FF",
        "BlueViolet": "8A2BE2",
        "Brown": "A52A2A",
        "BurlyWood": "DEB887",
        "CadetBlue": "5F9EA0",
        "Chartreuse": "7FFF00",
        "Chocolate": "D2691E",
        "Coral": "FF7F50",
```

```
        "CornflowerBlue": "6495ED",
        "Cornsilk": "FFF8DC",
        "Crimson": "DC143C",
        "Cyan": "00FFFF",
        "DarkBlue": "00008B",
        "DarkCyan": "008B8B",
        "DarkGoldenRod": "B8860B",
        "DarkGray": "A9A9A9",
        "DarkGreen": "006400",
        "DarkKhaki": "BDB76B",
        "DarkMagenta": "8B008B",
        "DarkOliveGreen": "556B2F",
        "DarkOrange": "FF8C00",
        "DarkOrchid": "9932CC",
        "DarkRed": "8B0000",
        "DarkSalmon": "E9967A",
        "DarkSeaGreen": "8FBC8F",
        "DarkSlateBlue": "483D8B",
        "DarkSlateGray": "2F4F4F",
        "DarkTurquoise": "00CED1",
        "DarkViolet": "9400D3",
        "DeepPink": "FF1493",
        "DeepSkyBlue": "00BFFF",
        "DimGray": "696969",
        "DodgerBlue": "1E90FF",
        "FireBrick": "B22222",
        "FloralWhite": "FFFAF0",
        "ForestGreen": "228B22",
        "Fuchsia": "FF00FF",
        "Gainsboro": "DCDCDC",
        "GhostWhite": "F8F8FF",
        "Gold": "FFD700",
        "GoldenRod": "DAA520",
        "Gray": "808080",
        "Green": "008000",
        "GreenYellow": "ADFF2F",
        "HoneyDew": "F0FFF0",
        "HotPink": "FF69B4",
        "IndianRed": "CD5C5C",
        "Indigo": "4B0082",
        "Ivory": "FFFFF0",
        "Khaki": "F0E68C",
        "Lavender": "E6E6FA",
        "LavenderBlush": "FFF0F5",
        "LawnGreen": "7CFC00",
        "LemonChiffon": "FFFACD",
        "LightBlue": "ADD8E6",
        "LightCoral": "F08080",
        "LightCyan": "E0FFFF",
        "LightGoldenRodYellow": "FAFAD2",
        "LightGray": "D3D3D3",
        "LightGreen": "90EE90",
        "LightPink": "FFB6C1",
        "LightSalmon": "FFA07A",
        "LightSeaGreen": "20B2AA",
        "LightSkyBlue": "87CEFA",
        "LightSlateGray": "778899",
        "LightSteelBlue": "B0C4DE",
        "LightYellow": "FFFFE0",
        "Lime": "00FF00",
        "LimeGreen": "32CD32",
        "Linen": "FAF0E6",
        "Magenta": "FF00FF",
        "Maroon": "800000",
        "MediumAquaMarine": "66CDAA",
        "MediumBlue": "0000CD",
        "MediumOrchid": "BA55D3",
        "MediumPurple": "9370DB",
        "MediumSeaGreen": "3CB371",
        "MediumSlateBlue": "7B68EE",
        "MediumSpringGreen": "00FA9A",
        "MediumTurquoise": "48D1CC",
        "MediumVioletRed": "C71585",
        "MidnightBlue": "191970",
        "MintCream": "F5FFFA",
        "MistyRose": "FFE4E1",
        "Moccasin": "FFE4B5",
        "NavajoWhite": "FFDEAD",
```

```
        "Navy": "000080",
        "OldLace": "FDF5E6",
        "Olive": "808000",
        "OliveDrab": "6B8E23",
        "Orange": "FFA500",
        "OrangeRed": "FF4500",
        "Orchid": "DA70D6",
        "PaleGoldenRod": "EEE8AA",
        "PaleGreen": "98FB98",
        "PaleTurquoise": "AFEEEE",
        "PaleVioletRed": "DB7093",
        "PapayaWhip": "FFEFD5",
        "PeachPuff": "FFDAB9",
        "Peru": "CD853F",
        "Pink": "FFC0CB",
        "Plum": "DDA0DD",
        "PowderBlue": "B0E0E6",
        "Purple": "800080",
        "RebeccaPurple": "663399",
        "Red": "FF0000",
        "RosyBrown": "BC8F8F",
        "RoyalBlue": "4169E1",
        "SaddleBrown": "8B4513",
        "Salmon": "FA8072",
        "SandyBrown": "F4A460",
        "SeaGreen": "2E8B57",
        "SeaShell": "FFF5EE",
        "Sienna": "A0522D",
        "Silver": "C0C0C0",
        "SkyBlue": "87CEEB",
        "SlateBlue": "6A5ACD",
        "SlateGray": "708090",
        "Snow": "FFFAFA",
        "SpringGreen": "00FF7F",
        "SteelBlue": "4682B4",
        "Tan": "D2B48C",
        "Teal": "008080",
        "Thistle": "D8BFD8",
        "Tomato": "FF6347",
        "Turquoise": "40E0D0",
        "Violet": "EE82EE",
        "Wheat": "F5DEB3",
        "White": "FFFFFF",
        "WhiteSmoke": "F5F5F5",
        "Yellow": "FFFF00",
        "YellowGreen": "9ACD32"
      },
```

This section deals with document styles. It has 3 sections: "pages", "front_page", and "toc". If a style is not set in a specific section it will fall back to what is specified in a more general section. For example, if a subsection of "document" does not specify "color" then it will fall back to the "color": "black" directly under "document".

```
      "document": {
        "pageFormat": "A4",
```

For the margins the suffix can be "cm" for centimeters, "in" for inches or "pt" for points. This value can also be specified as a JSON number in which case it is in points.

```
        "leftMargin": "2.54cm",
        "rightMargin": "2.54cm",
        "topMargin": "2.54cm",
        "bottomMargin": "2.54cm",

        "color": "black",
        "background": "white",
        "family": "HELVETICA",
        "size": 10,
        "style": "Normal",
```

The section number offsets allows for changeing the position slightly for the section numbers when they are enabled.

```
         "sectionNumberYOffset": 2.0,
         "sectionNumberXOffset": -10.0,

         "image": {
            "imgScalePercent": 60.0,
```

The alignment can be either "LEFT", "MIDDLE" or "RIGHT". Note that if "imgX" and "imgY" is set, then this does not apply.

```
            "imgAlign": "LEFT",
            "imgRotateDegrees": 0.0,
```

If "imgFlow" is set to true then text will flow around the image. Basically what happens is that when text is about to overwrite the image then it is moved right to after the image and continues from there. To get this effect you can place an image in the middle of a paragraph and it will flow around the image.

```
            "imgFlow": false,
```

This margin is always in points and determins the space to reserve to the left and right of an image when "imgFlow" is set to true. This to avoid having text and image exactly side by side with no space, since that tends to look strange.

```
            "imgFlowMargin": 4.0,
```

These 2 allows you to override the position of an image on the page. This works best in conjunction with "imgFlow". Also note that this does not specify a specific image! If you specify this directly under "document" then all images on the page will be rendered over each other at this coordinate! So it makes much more sense to use this feature in conjunction with a div, in which you also place the image. I'm only putting it here now to show its association with "imgFlow".

```
            "imgX": 127.0,
            "imgY": 430.0
         },

         "pages": {
```

The style value can be any of NORMAL, BOLD, ITALIC, and UNDERLINE. UNDERLINE can be used in conjunction with the other, comma separated. Example ITALIC,UNDERLINE.

```
           "block_quote": {
             "style": "ITALIC",
             "color": "mddgrey"
           },
           "h1": {
             "size": 20,
             "style": "BOLD"
           },
           "h2": {
             "size": 18,
             "style": "BOLD",
```

"underlined" draws and underline under the heading from left margin to right margin. The "underline_offset" is how many points below the text to draw the line. In previous versions this was called "hr".

```
            "underlined": true,
            "underline_offset": 3.0
          },
          "h3": {
            "size": 16,
            "style": "BOLD"
          },
          "h4": {
            "size": 14,
            "style": "BOLD"
          },
          "h5": {
            "size": 12,
            "style": "BOLD"
          },
          "h6": {
            "size": 10,
            "style": "BOLD"
          },
          "emphasis": {
            "style": "ITALIC"
          },
          "strong": {
            "style": "BOLD"
          },
          "code": {
            "family": "COURIER",
            "size": 9,
            "color": "64:64:64",
```

If "preformattedWordWrap" is set to true, then "code" style text will not entirely be left as is, but will wrap around to a new line if the text does not fit within the margins, and this will be with an indent matching the "code" text plus some more indent to show that it is a continuation of the previous line. Depending on the text this sometimes works well, sometimes less than well.

```
            "preformattedWordWrap": false,
```

When "boxed" is set to true then a filled box is drawn below the text. It ranges from the left margin to the right margin for multiline (indented) "code" text. For `text` variant only the text is boxed.

```
            "boxed": true,
            "boxedColor": "#f8f8f8"
          },
          "anchor": {
            "color": "128:128:128"
          },
          "list_item": {
          },
```

This is also new in 2.0.0 and sets the thickness and color of a horizontal ruler.

```
          "horizontal_ruler": {
            "thickness": 0.5,
            "color": "grey"
          }
        },

        "divs": {
          "mdd-example": {
            "color": "white",
```

```
            "background": "black",
            "block_quote": {
              "family": "COURIER",
              "color": "120:120:120",
              "background": "10:11:12"
            }
          },
          "center-page5-image": {
            "imgX": 127.0,
            "imgY": 430.0
          }
        }
      },

      "front_page": {
        "color": "0:0:0",
        "background": "255:255:255",
        "family": "HELVETICA",
        "size": 10,
        "style": "NORMAL",

        "image": {
           "imgScalePercent": 60.0,
           "imgRotateDegrees": 0.0
        },

        "title": {
          "size": 25,
          "style": "UNDERLINE"
        },
        "subject": {
          "size": 15
        },
        "version": {
          "size": 12,
        },
        "copyright": {
        },
        "author": {
          "size": 12,
        }
      },

      "toc": {
        "color": "0:0:0",
        "background": "255:255:255",
        "family": "HELVETICA",
        "size": 9,
        "style": "NORMAL",
        "h1": {
          "style": "BOLD"
        },
        "h2": {
        },
        "h3": {
        },
        "h4": {
        },
        "h5": {
        },
        "h6": {
        }
      }
    }
```

# Settings / Options

The options for each generator are represented by a JavaBean model using OptionsManager annotations. Independent of how you run a generator it is the same options model that is used the only difference is in how it is populated. If it is from code you just use setter methods to set values. If it is from the command line jar then each option name is prefixed with -- and passed as command line argument. In this case it is OptionsManager that will populate the model from arguments. If it is from the maven plugin then each option is set in the pom with xml tags of the same name as the options. In this case it is maven that populates the options.

The options will be described here in general, not centric to any way of running.

**(R)** after an option means it is required.

**(O)** after an option means it is optional.

Note that values of **Boolean** type should have a value of either "true" or "false". They do need a value when specified from command line! Just a *--firstLineParagraphIndent* will not work!

## *Common options*

### generator (R)

Specifies the generator to run. Current valid values are:

• pdf
• html
• markdown

### parser (O)

Selects the parser to run. Valid values are:

• markdown
• byext[ension] *This is the default value!*

The latter selects parser based on extension of file being parsed. This is in general a good idea to use since there are currently both a markdown parser and a javadoc parser. By using *byext* it ispossible to pass both .md and .java (and .groovy) files. Each parser is registered as a standard Java service and loaded with ServiceLoader. Each parser also provides which file extensions it recognizes. This is how a parser is resolved.

### inputPaths (R)

A comma separated list of files and paths. A path can look like this: `MavenPlugin/docs/.*.md`. A path supports regular expressions.

Files will be parsed in the order they are specified. When regular expressions are used to include multiple files the order is unspecified.

### parserOptions (O)

This is a comma separated list of `name=value`.

#### JavaDoc2MDParser options

Just setting **markdownJavadoc** to any value will make it take the javadoc text as markdown and parse that also.

### MarkdownParser options

This parser currently has no options.

## *PDFGenerator options*

### resultFile : String (R)

The path to the PDF file to write.

### rootDir : String (O)

A root dir to make image paths relative to.

### title : String (O)

The title of the document. This is used if **generateTitlePage** is set to true.

### subject : String (O)

The subject of the document.

### titlePageImage : String (O)

Put an image on the title page. Format: <path/URL>:x:y

### author : String (O)

The author of the document.

### version : String (O)

The version to put on the title page. Must be specified to be rendered!

### copyright : String (O)

The copyright message to put on the title page. Must be specified to be rendered!

### hideLinks : Boolean (O)

If true then links are not rendered as link the link text will be rendered as plain text.

### unorderedListItemPrefix : String (O)

What item marking to use for unordered lists. Default is '- '.

### mss : String (O)

This specifies the path to an .mss file to use for setting fonts and colors and image styling of the generated document.

### generateSectionNumbers : Boolean (O)

If true all chapters and sections will be numbered. This was the only option before version 1.4.

### generateTOC : Boolean (O)

This generates a table of contents. Default is false!

## generateTitlePage : Boolean (O)

This will generate one first page with a title, version, author, and copyright. Default is false.

## help (Only from command line!)

Shows help.

## Comment block annotation setting of options

The PDF generator have a special feature to be able to set options via an annotation in a comment block. The annotations look like this:

```
<!--

    @PDF<option name>(<option value>)
    @PDF<option name>("<option value>")

-->
```

The following annotation options are available:

• @PDFTitle(title)
• @PDFSubject(subject)
• @PDFKeywords(keywords)
• @PDFAuthor(author)
• @PDFVersion(version)
• @PDFCopyright(copyright line)
• @PDFPageSize(size)
• @PDFHideLinks(true/false)
• @PDFUnorderedListItemPrefix(prefix)
• @PDFGenerateSectionNumbers(true/false)
• @PDFGenerateTOC(true/false)
• @PDFGenerateTitlePage(true/false)
• @PDFTitlePageImage(imageref)

Put this comment block at the top of the document! The optoins provided this way will not have an effect until the comment block have been processed by the generator, and the annotations found. Thereby it is also theoretically possible to change options further down the document. This should be seen as a side effect rather than a feature!!

# *HTMLGenerator options*

## resultFile : String (R)

Where to write the result.

## inlineCSS : Boolean (O)

If true then the css will be included in the generated HTML.

## css : String (O)

The path to a CSS file.

## primitiveHTML : Boolean (O)

When true very primitive HMTL will be generated. This will work for rendering with JEditorPane.

## makeFileLinksRelativeTo : String (O)

The path file links should be relative to.

## help (Only from command line!)

Shows help.

## *MarkdownGenerator options*

### resultFile : String (R)

Where to write the result.

### makeFileLinksRelativeTo : String (O)

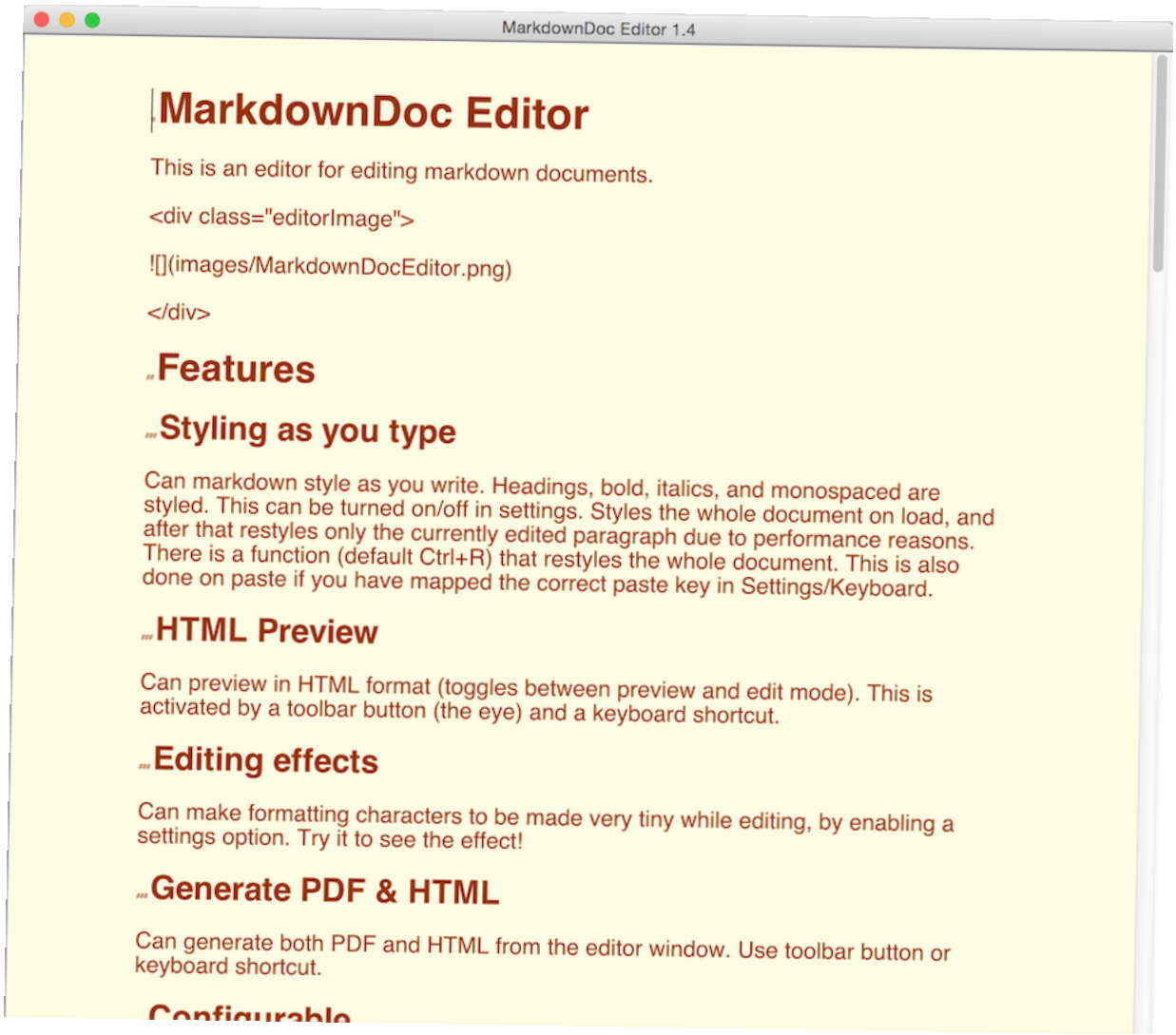
The path file links should be relative to.

### help (Only from command line!)

Shows help.

# MarkdownDoc Editor

This is an editor for editing markdown documents.




## *Features*

### Styling as you type

Can markdown style as you write. Headings, bold, italics, and monospaced are styled. This can be turned on/off in settings. Styles the whole document on load, and after that restyles only the currently edited paragraph due to performance reasons. There is a function (default Ctrl+R) that restyles the whole document. This is also done on paste if you have mapped the correct paste key in Settings/Keyboard.

### HTML Preview

Can preview in HTML format (toggles between preview and edit mode). This is activated by a toolbar button (the eye) and a keyboard shortcut.

### Editing effects

Can make formatting characters to be made very tiny while editing, by enabling a settings option. Try it to see the effect!

# Undo / Redo (as of 1.4)

Attaches an UndoManager to the document model. Ctrl-Z does an undo, and Ctrl-Y does a redo for all platforms except Mac, which uses Meta-Z for undo, and Shift-Meta-Z for redo.

Do note however that if you enable styling as you type in settings, then the styling actions also gets recorded by the UndoManager! So the first Ctrl-Z might just undo the styling, and the next Ctrl-Z does what you expected it to do. I have currently not found a way around this.

# Generate PDF & HTML

Can generate both PDF and HTML from the editor window. Use toolbar button or keyboard shortcut.

# Configurable

The settings dialog allows you to configure almost anything/everything:

- All keyboard shortcuts.• Don't write the keyboard shortcut in text, just press the keyboard shortcut you want to set.
- Margins.
- Editor font.
- Monospaced font.
- Preview font.
- Font sizes.
- Background color.
- Text color.
- Toolbar variant to use.

# Load file by drag & drop

Instead of using the GUI open dialog you can just drag and drop a file in the editor to edit it.

# Special preview drag & drop feature

While in preview mode, drag and drop a markdown file on the preview window to have it formatted and displayed. This does not affect the edit buffer in any way. Exiting preview mode will bring you back to whatever you have in the editor, and previewing again will preview the editor content.

But by just opening an empty editor and entering a blank preview you can quickly read multiple markdown documents formatted by just dropping them on the window.

# Fullscreen support

When in fullscreen mode then the settings popup will still popup to the right, but on top of the editor window, so it will still work in that mode.

## Mac OS X

When you run this editor on a Mac with Lion+ you will get a litte double arrow in the right corner of the window titlebar, or with Mavericks+ it will be the green dot, which will bring upp the editor window in fullscreen.

## Windows (10)

On Windows10 pressing the square button on the top right side of the window will enter some kind of fullscreen.

## MarkdownDocEditor.app

The editor is also available in MarkdownDocEditor.app format. The build plugin that creates this .app packaging does however not support passing on arguments to the app when run this way. That means selecting a markdown file and doing "open with" will fail. It will always open the file chooser for you to select the file(s) to edit.

Also note that since this .app is not signed, Mac OS X Mavericks and upp will not allow you to run the app if you do not open upp for running all types of apps in the security settings.

Due to GUI bugs in the editor component of earlier versions of Java, Java 1.8 or higher is required to be installed on your system for the app to be able to run, assuming you are using the version I'm providing for download in the README.md document. If you checkout the source and build yourself then it will require Java 1.6 and upp. But be warned: You will be annoyed if you use Java lower than 1.8!

# Running

Can be run with java -jar or double clicked on. If you are using Windows 7 or 8 take a look at this page: http://johann.loefflmann.net/en/software/jarfix/index.html (http://johann.loefflmann.net/en/software/jarfix/index.html).

The executable jar have the following name: MarkdownDocEditor-n.n.n-App.jar

One or more files or directories can be specified as arguments. For a directory all markdown files found in the directory and subdirectories will be loaded. As said above this does not apply if you are running the .app version.

# Requirements

This requires Java 1.8+!

# Functions

This section documents the different functions of the editor, and how to trigger the function.

The images are the toolbar icon for the function. Not all functions install themselves in the toolbar.

## Bringing upp the toolbar

Move the mouse to the top of the editor window and the toolbar will automatically popup. Move the mouse down again and it will go away.

## Save file(s)

Default key: Ctrl+S. This is changeable in the settings.

This saves all open files that have been modified and not saved. A small pupup appears for a short while in the upper left corner of the window to indicate how many files were saved. If it the number is 0 then there were no modified files needing save.

## Open file

Default key: Ctrl+O. This is changeable in the settings.

This opens a file chooser to select one markdown file to open. The opened file will be selected for editing in the window.

This function is kind of unneccesarry in this version, but I decided to leave it in anyhow. It differs slightly from Open/Create. It is likely to go away in future versions.

In addition to markdown files the open function will also allow opening the same .fs files as the maven plugin can use. In this case all references to markdown files in the .fs file will be opened.

## Open / Create

 Default key: Ctrl+N. This is changeable in the settings.

This opens a file chooser where you can also enter a filename in the chooser dialog. Here you can navigate to a directory, and then enter the name of a new file that will then be created and opened. In this file chooser you can alternatively navigate to a directory and then select one or more existing files and have all selected files being opened.

Since there can be more than one file, no file is set as the current edited in the window. You have to bring up the list of open files and select one of the newly added files to edit it.

The exception to this is when you have started the edtior without any files, which will triger this file chooser then one of the selected files will become the edited file since there always have to be one file in the editor.

In addition to markdown files the open function will also allow opening the same .fs files as the maven plugin can use. In this case all references to markdown files in the .fs file will be opened.

## List of open files popup

 Default key: Ctrl+1. This is changeable in the settings.

This will popup a window to the left of the editor window, or to the left on top of the edtitor window if fullscreen. It will list all open files with filename, and some starting file content to make it easier to identify. This idea was borrowed from other tools. If there are more open files than fits vertically the popup is scrollable vertically. Click on any file to set that file as the currently edited. The popup window till not close until you click in the editor window.

Keyboard use in the popup are currently not supported.

## Insert heading

 Default key: Ctrl+T. This is changeable in the settings.

This just adds a # character which is the markdown heading character. Insert as many as the heading level you want, max 6.

## Insert bold

 Default key: Ctrl+B. This is changeable in the settings.

This adds 4 '_' characters with the cursor placed between the first 2 and the last 2. 2 underscores before and after makes bold text in markdown. 2 asterisks before and after does the same thing, but the editor uses underscores for this specific help function.

## Insert italics

 Default key: Ctrl+I. This is changeable in the settings.

This adds 2 '_' characters with the cursor placed between them. 1 underscore before and after makes italic text in markdown. 1 asterisk before and after does the same thing, but the editor uses underscores for this specific help funciton. Asterisks also means other things in markdown so underscores in this case is less confusing.

## Insert list

 Default key: Ctrl+L. This is changeable in the settings.

This adds and asterisk and a space which is how you make a list entry for an unordered list in markdown. Do note that it is also possible to make a numbered list, in which case you replace the asterisk with a number like 1. See the markdown reference section of this document for more information.

## Insert quote

 Default key: Ctrl+Q. This is changeable in the settings.

This inserts a '>' character and a space which is how you make quoted text in markdown.

## Insert image

 Default key: Ctrl+M. This is changeable in the settings.

This function will open a small popup window where you can enter 3 pieces of information: 1) An alt text. 2) A URL to the image. 3) An image title. Only the URL is required. When you press the "Insert" button in the popup window, then the image reference will be inserted into the text in markdown format: `![Alt text](url "title")`.

## Insert link

 Default key: Ctrl+N. This is changeable in the settings.

This function will open a small popup window where you can enter 3 pieces of information: 1) The link text. 2) The link URL. 3) A link title. You should provide a link text and an URL as minimum. When you press the "Insert" button in the popup window, then the link will be inserted into the text in markdown format: `[link text](url "title")`.

## Preview

 Default key: Ctrl+F. This is changeable in the settings.

This will format the markdown in the editor into HTML and show it in readonly mode instead of the editable content. To go back to editing again do a Ctrl+F again or use the toolbar button. Do note that while in preview mode it is possible to drag and drop markdown files into the window to have them previewed. This does not affect what you are editing in any way. When you go back to edit mode again your edited text will be there and a new preview will preview that text.

*Please also note that the preview HTML rendering is done by the Java GUI library (Swing) component JEditorPane. This is far from an optimal HTML renderer! It can make things look strange sometimes. It also insists on indenting the beginning of every code block. If anyone knows of a free, open source, swing compatible HTML renderer, that is better please let me know.*

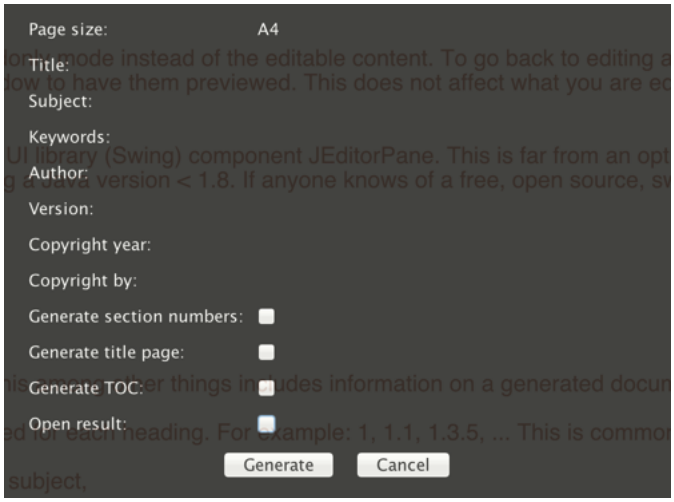
# Generate PDF

Default key: Ctrl+P. This is changeable in the settings.

This will first open a file chooser to select target PDF file to generate to. Then a popup window with meta data for the PDF generation will open.

Press the "Generate" button to actually generate the PDF document.

The choices are: (Note that pic is old, keywords no longer exists!)




## Title

This is the title of the document. This will be shown on the front page.

## Subject

This is an optional subject / subtitle. This will be shown on the front page.

## Author

The author of the document. This will be shown on the front page.

## Version

The current version of the document. This will be shown on the front page.

## Copyright year

The year of copyright. This will be shown on the front page in the format: "Copyright (C) {year} by {by}".

## Copyright by

The one holding the copyright.

## Generate section numbers

When this is selected numbers are generated for each heading. For example: 1, 1.1, 1.3.5, ... This is common for professional documents.

## Generate title page:

This will produce a first page with a document title, subject, author, version and copyright.

## Generate TOC

This will generate a table of contents. This will come after the title page if that is generated, but before the document.

## Open result

If this is selected then the generated PDF will be opened by the system default PDF viewer.

# Generate HTML

Default key: Ctrl+H. This is changeable in the settings.

This will first open a file chooser to select HTML file to generate to. Then a popup window will be opened containing meta data for generation of the HTML.


## Inline CSS

If you select this option then the CSS file you point out will be included within the generated HTML file.

## CSS file

This is the CSS file to use. Write a path to the CSS file or use the "Select" button to open a file chooser. This is optional and can be skipped, but the resulting HMTL can be rather boring if you don't provide a CSS.

## file: links relative to

This is a path that file: links in the document isrelative to. This is used to resolve local filesystem images.

## Open result

If this is selected then the generated HTML will be opened by the system default browser.

# Setting

Default key: Ctrl+E. This is changeable in the settings.

This opens the settings popup window where you can configure keys, margins, colors, etc.

# Restyle document

Default key: Ctrl+R. This is changeable in the settings.

This will force a complete restyling of the whole document.

# Restyle on paste

Default key: Ctrl+V. This is changeable in the settings.

This also forces a restyle of the document, but when paste of text into the document is done. For this to work it must be mapped to the same key as is used for paste. On windows and linux it is Ctrl+V, on Mac it is Cmd+V. This function can be disabled by setting the key to something else than the paste key.

# Goto next open file

This is an alternative to the above function and allows you to jump around the open files. Each time this function is triggered the editor window will switch content. Internally the editor just keeps a list of all open files, and this just jumps to the next file in that list until the last file is reached and it jumps to the first instead. This function will also show the name of the current "jumped to" file at the bottom of the window just like when you move the mouse up to the toolbar area.

Note that this function has no default key configured and must be set in the settings window before it can be used. On Mac Alt+Tab works since it is not used by the system, on Windows and Linux you probably need some other key combination.

This function is new in version 1.4.1.

## If you're on a Mac

If you are on a Mac you might want to change the keyboard mappings to use Cmd rather than Ctrl. Do note however that Cmd+H and Cmd+Q are really nasty on Mac OS X! Since these keys immedialtely kills the app these keys are impossible to set in the first place, but you will loose other unsaved settings when you try.

## Currently Missing

Fancy functions like search and replace.

## Laptop power warning

The markdown styling as you type in the editor do pull some CPU since basically the whole paragraph needs to be reformated on every key. My Mac Book Pro marks this app as a heavy energy user. To minimize battery drain you can turn off the styling as you type in the settings.

## Bugs

### By me

All *known* bugs have been fixed.

### By Oracle

This editor uses the standard Swing component JTextPane. This is unfortunately not an optimal component. Specially for styling it gets slow for large documents. In earlier versions of Java 7 this component had a word wrap problem when deleting text either using backspace or cutting text. In that case it rerendered the text screwing up the format until new text was entered again. *As of Java 8 this bug is fixed*, but other new bugs have been added. They are however smaller and don't occur so often.

Unfortuneately 1.8_u92+ is really screwed and old bugs are back in JTextPane. Anything can happen, most often text below where you are currently writing will be screwed formatwise, even the line you are writing on. Words no longer wrap and text is lost to the right. A Ctrl/Cmd-R fixes this until you start typing again. The whole document from top to bottom needs to be restyled in one go for things to be correct. MarkdownDoc Editor however only restyles the paragraph you are writing in as you are writing. This because restyling the whole document is far too slow!

sometimes when the JTextPane is opened the pane will not render at all! Just increase the width of the window util text appears. Then save so that the window size for that file will be remembered. I have one and only one document for which this happens and I cannot tell what it is that causes the problem. This could be a mac only problem.

### Other

On Windows 7 generating a PDF within the editor throws a NullPointerException! This does not happen on Mac OS X, Linux, nor Windows 10. I see this as a Windows 7 bug.

# The mddoc file type

There is a special file type that describes a complete document in any or all of the 3 output formats. It has the extension of *.mddoc*. It is really a properties file with *key: value* entries.

A path to an *.mddoc* file can be specified as only argument to command line variant, or as only file in `<inputPaths>...</inputPaths>` section in maven plugin (no other options/parameters are needed then) to produce output documents as described by the *.mddoc* file.

## *.mddoc format (myfile.mddoc)*

```
# --- Generators to run ---
generate.pdf: true
generate.html: true
generate.markdown: true

# A comma separated list of paths to sources. A .fs file can also be
# specified as an input file in which case it is read for a further
# set of files to parse. The order of the specified files are important.
inputPaths: docs/intro.md,docs/install.md,docs/usage.md,docs/appendix.md

# --- PDF ---

# The name of the file to produce.
pdf.resultFile: MyDoc.pdf

# The page size. For example:A4, LETTER Optional. Default: A4
pdf.pageSize: A4

...

# --- HTML ---

# The name of the file to produce.
html.resultFile: MyDoc.html

# The path to the css file for the generated html file. Required.
html.css: css/my.css

...

# --- Markdown ---

# The name of the file to produce.
markdown.resultFile: MyDoc.md

# This affects links and images. When specified the resulting file: URLs in the
# result will be relative to the path specified by "path" if the absolute path
# of the URL starts with the specified path. If a plus sign (+) and a prefix
# path is specified it will be prefixed to the final URL. Optional.
markdown.makeFileLinksRelativeTo: path[+prefix]
```

As you can see pdf options are prefixed with "pdf.", html options are prefixed with "html.", and markdown options are prefixed with "markdown.". After the prefix are the same options as documented under the "Options / Settings" section.

# Version history

About versions, they are hell! After personal experience of having different versions for each module / produced jar which was close to impossible to keep track of which was compatible with which, I decided to go with just one and the same version for all modules of the tool. This has the side effect of changes in any submodule, even the editor, which probably not everyone uses, will change the version for all even though no changes have been done for some modules. What have changed for each version is documented below so that you can determine if upgrading to the latest version is wanted/needed or not.

## *2.0.1*

This is yet another editor only update. The popup window for selecting loaded/open file to edit were quite bad. This have now been completely redone. Now a popup window pops upp to the left of the editor window and lists all open files, and some starting text from the file to make it easier to identify. An idea I borrowed from other tools.

## *2.0.0*

PDFBox is now used instead of iText to generate PDF. This required some non backwards compatible changes so thereby the version is bumped to 2.0.0. Note that the incompatibilities are small, and most likely this version will work without changes for many.

- Keywords are gone.
- Footer is no longer supported. Can be added if enough wants it. I have had no use for it myself.
- pageSize is no longer an option, but an MSS setting. This was a decision I made due to now being responsible for all rendering on the page and thus having more control over things like margins (now also settable in MSS), etc.
- There is a difference in image types handled. iText supports JPEG, JPEG2000, GIF, PNG, BMP, WMF, TIFF, CCITT and JBIG2 images. I can't find a clear list of image types supported by PDFBox (which in general is bady documented, I had to use Stack Exchange a lot!), but from MarkdownDoc:s perspective those image types supported by AWT are supported. The image types supported by PDFBox, not using AWT, like TIFF are not supported since that API only allows loading images from disk! This works badly together with URLs. Yes, it would be possible to download an image to disk first, then pass it to the API, and then delete it locally or cache it for later reuse. But I decided agains that now.
- The "hr" MSS value for headings have been renamed "underlined", which is by far more clear. This has nothing to do with anything else, just a decision I made since other things have been changed, why not fix this also. I also added an "underline_offset" to set how many points below the text to draw the underline.
- Page size is no longer supplied as an option! This is now set in the MSS file used. Default is A4. Margins now defaults to what I can determine from googling is the default for A4: 2.54 cm. These can also be set in MSS.
- I no longer use labels like "Author:" (on front page) or "Page" before page number, etc. I don't miss them, and it does not look strange without them IMHO. This also means the "label" settings for these texts are not needed.
- `<!-- @PageBreak -->` now works! Previously when using iText, iText ignored all my tries to force a page break!

I added some features in MSS:

- Boxed. Current default.mss uses this for *code* style. A box of choosen color is rendered below text.
- Positioning of images on page.
- Allowing text to flow around images. When an image is added to a page a "hole" the size of the image is defined in the page, and any text rendering will skip the hole and continue after it. This is optional behavior.
- Setting page size (A4, LETTER, etc).
- Overriding default page margins.

See the MSS section of the documentation for more info.

The reason for this change is that I discovered that iText is using a GPL license! Now you might think, "What the heck is he talking about ?, the GPL license text have been included in the docs all the time!". Well, that information is generated automatically by another of my tools: CodeLicenseManager. It finds all licence information in pom:s and include license texts. I haven't looked that closely at what licenses are included. Obvioulsy I should have. It however hit me this summer and I decided to go looking for antother Java PDF library, and found Apache PDFBox. PDFBox is of course under the very sensible "Apache Software License 2.0", the same license I'm releasing MarkdownDoc under. I suspect that the way the GPL is used today was not the intention of Mr Stallman. The GPL nowmore tends to make non free software look free, and that is exactly how iText is using it.

PDBox however have some pluses and some minuses:

+

Lower level, closer to PDF. This gave me much more flexibility and I can now generate everything only once since I now can insert the table of contents at the top of the document after generating the contents, which is needed to get the page numbers for the table of contents. With iText I had to make a dummy generation to a null stream first, just to get page numbers.

Since it is so low level it does not have the type of bugs that iText have. Now all bugs should be mine :-). That is good since then I can do something about them.

It was now easy to render boxed backgrounds for preformatted text. I always wanted to do that, but I could not figure out how to do it with iText since iText never gave me the coordinates of the text. Now I have full control over the coordinates of everything.

-

PDFBox is slower than iText especially when images are used.

PDFBox unfortunately uses AWT for handling most images! This has consequences! Whenever PDFBox is dealing with a PNG, JPG, etc a small window is opened. It is of course closed again when it is done with the image handling. But if run on a server to generate som PDF report then the server process needs access to an X server if running on a unix system! This is however only if images are used.

## JDK Level

This version is built with JDK 1.8! The Groovy code might still produce 1.5 compatible bytecode, but the maven plugin is written in Java and thus requires 1.8+ to run. The editor also have less bugs when run with 1.8. 1.7 went 6 feet under over a year ago, so you shouldn't be using anything lower than 1.8 anyhow.

# *1.4.4*

## Editor

The settings poppup is now popped up to the right of the window, not over it, unless in fullscreen mode then the popup will still be to the right, but on top of the editor window obviously :-). This makes it easier to se the result when playing with settings.

# *1.4.3*

## Library

Implemented a suggestion from Mikhail Kopylov that also allows images to be referenced relative from the .md file they are part of.

Added information about options for referencing images in *MarkdowndDoc* under the *Markdown Reference* section.

## 1.4.2

### Editor

The editor has been updated:

- The popup window of all open files is no longer accessable by moving the mouse pointer to the left side of the window. This was a really bad idea! It was very easy to unintentionally trigger this popup. This function has now gotten an icon and been added to the toolbar instead. It already had a keyboard shortcut before and still does.
- All toolbar icons have been redone.
- It is now also possible to open the same .fs files as supported by the maven plugin. All markdown documents referenced in the .fs file will be opened in the editor. If an .fs file also points out a java source file for javadoc parsing it will be ignored. Only markdown documents are opened.

## 1.4.1

Only fixes in editor!

- The popup windows now only popup over the editor window. The fullscreen popups worked badly on different platforms which reserves different parts of the screen.
- Disabled rounded corner popup windows since they also worked with different quality on different platforms.
- Added editor function to *Alt-Tab or something* around opened files in the editor. For this to work at all you need to open configuration and set a keyboard compination that triggers this. There is no failsafe default that works on all platforms. The config is called "content switch keyboard shortcut".

## 1.4

- Added support for what I call *Markdown Style Sheet* or MSS for short. This is only applicable to PDF generation. For HTML there is CSS, and generating CSS from the MSS is a bad idea. The MSS is relatively simple and JSON based.• It supports ttf, otf, and any other format supported by iText for external fonts.
  - It allows for image configuration like scaling, rotating, and alignment. Before all images were alinged to the left. Now they can be alingned to the left, middle, or right. In previous versions all images was scaled to 60 percent due to iText rendering images very much bigger than any other image viewer (that I have at least). This scaling can now be set with MSS.
- Added support for `<div class="..">...</div>`. This tool is mainly for writing documentation and generating PDF, but I wanted to add more flexibility for generating HTML pages also. Even though you probably want to keep a common style throughout a document, I did add div support to MSS. Divs within divs inherit styles upward. This was relatively simple to do. Note that the "Options / Settings" section uses a div with slightly different formatting than the rest of the document. Each option is a level 3 heading (H3) which is why it is part of the TOC, but styled with a smaller font with a different color.
- Added possibility to also have an image on the title page.
- Added annotations within a comment block. Most of the options for the PDF generator can now be specified with annotations in the document. For example `@PDFPageSize("A4")`. This means for example that the title page can be part of the document. This comment with annotations should preferrably be the first thing in the document. The reason for the very Javaish format of the annotation is that it is explicit enough to not be accidentally and unwantedly matched.
- Added labels in options for all previously hardcoded text strings in PDFGenerator. It should now be possible to completely generate a document in a different language than English. These can also be set with comment annotations as mentioned above.
- Added Undo / Redo to editor. Swing apparently provides support for this for a JTextPane/JEditorPane, you just have to register an UndoManager. **However**, it reacts on all

changes, including styling. Since styling is applied afterward, and on the whole paragraph since it is not only about the current character, this is also registered as a change in the undo manager. So undos will undo styling also. So after undoing what you wanted to undo, do a Ctrl/Cmd+R to fix styling again.
- Editor updated quite a lot!• All *known* bugs fixed.
  - Now handles multipe files in one editor window. It actually only supports one window now, but you switch between the open files by moving mouse to the left window egde which will popup a list of all open files and you just click on the one you want to work on.
  - It is possible to specify a directory as input to editor. In this case it will scan the directory for markdown files and open all found. It is actually possible to specify multiple directories just as it is possible to specify multiple files.
  - When the toolbar is shown at the top of the window, the name of the current file is also shown att the bottom of the window.
  - The editor GUI has gone though some cosmetics.

The addition of MSS made huge changes to the code. To be as backwards compatible as possible, the defaults for the MSS settings are as things looked before. There is also a *default.mss* file that gets used if you don't supply your own. This has settings that mimics the previous styles.

Also note that the PDF UserGuide now shows off the new features, mostly for that purpose :-).

## 1.3.9

Only bugfix in editor when generating HTML directly from editor, which caused an NPE.

## 1.3.8

Bad internal version dependencies in well ... probably from version 1.3.4 up to 1.3.7. The markdowndoc-maven-plugin were using a too old (hardcoded!!) version of markdown-doc-lib, which is the core of MarkdownDoc! It was pointing to version 1.3.3. This means that fixes in 1.3.4 and 1.3.5 were not available when maven plugin was used! It now uses ${project.version}. The command line jar and the editor have had the correct version dependency.

Very sorry for this!

## 1.3.7

- Bugfixes in the maven plugin.
- The maven plugin also no longer has any runtime dependency on CodeLicenseManager which is a build only plugin, something maven does not really distinguish.
- Includes a pull request submitted by both komarevsky and iorixxx that fixes an XML error in an example in the user guide. Thanks for seeing that and submitting pull requests!

## 1.3.6

Bug fixes in MarkdownDocEditor:

- Preformatted styling should now behave correctly.
- Preformatted font (monospace) settings now work. Also defaulted font size of monospace to 14 rather than 16.

## 1.3.5

What I did not mention in the information for version 1.3.4 is that the editor was converted from Java to Groovy. Here I apparently ran into a Groovy gotcha: What looked to be a member reference were actually a property reference to the same method that tried to reference member. In this case it was an anonymously implemented interface with a getter whose implementation tried to reference the outer class member of same name as getter property, and got the property rather than the member causing a never ending loop resulting in java.lang.StackOverflowError.

This affected only generating of PDF and HTML. The error occured after writing generated output, but before opening the generated output (when told to do so by checkbox setting). This problem is now fixed by this version and is the only thing that differs from previous version.

## 1.3.4

Fixed a bug with relative path for images using *PDFGenerator* reported by Maher Gamal. There are now 5 ways to specifiy paths to images for PDF:

1. Absolute path
2. Relative to current directory.
3. Relative to markdown document.
4. Relative to resulting PDF document.
5. Relative to a supplied root dir. This can now be specified in the PDF generator options. If using the library, passing rootDir will override the options rootDir.

These paths will be automatically resolved.

## 1.3.3

Ironed out all *known* bugs in editor.

## 1.3.2

Added markdown formatting as you write.

## 1.3.1

Bug fixes. Monospaced font now rendering correctly.

Deleting text with backspace have strange effects on text layout. That is, the place where a senetence is broken to the right and moved down to the next line keeps moving around while deleting text, in some completely different paragraph! This is entirely handled by JTextPane. I have tried to find a way to intercept the delete key and handle delete myself, but I have not been successful in finding a way to do that if it is even possible. Continuing writing new text after deleting text seems to restore the layout. This oddity has no effect on the final text, it is just the layout while editing that is affacted. You will also only see this if you write paragraphs as one block of text that wraps around into multiple lines without pressing return until the end of the paragraph.

## 1.3

Made big changes to the editor, finally making it into what I want, with some markdown formatting as you write, and far more configuration in settings dialog, which have also been redone.

Bug fixes.

## 1.2.10

Added support for <, >, and &.

## 1.2.9

Added markdown file reading feature by allowing markdown files to be dropped on the editor in preview mode, in wihch case the dropped file will be formatted and displayed without changeing the content of the editor. Exiting preview and doing a preview again will again preview the editor content.

## 1.2.8

Headings can now **not** be more than one line (not include LF/CRLF). Before they were treated like paragraphs. This to be more compatible with other Markdown tools and Markdown documents.

## 1.2.7

Added settings for specifying top, bottom, left, and right margins in editor. Please note that I've been a bit lazy here. The sizes are in pixels, not characters/lines!

## 1.2.6

Added the new *.mddoc* format, which makes command line usage easier, but it is also supported by the maven plugin and the library has a utility that completely handles this format.

Added a Java Swing based editor for editing markdown with support.

## 1.2.5

Added *parserOptions* now used by JavadocParser to markdown parse javadoc comments if markdownJavadoc=true is provided. The Parser API is thus also updated to take a Properties object for the parser options.

## 1.2.4

Added *makeFileLinksRelativeTo* option for HTMLGenerator and MarkdownGenerator mostly to be able to manipulate *file:* references to images in the generated result so that the image paths still work in source when editing with a markdown tool and is still correct when generated to a different path.

## 1.2.3

If image paths are not absolute and not http referenced then they are now looked for relative to the source markdown file first, and then the are looked for relative to the result file as before. This makes it easier to generate a big document for a whole project containing several subproject with local makdown documents and referenced images. The image reference can still be relative to the subproject local markdown file.

## 1.2.2

Added support for non breaking space (nbsp) to be able to indent text. This is one more exception to no html pass through.

# Simple Markdown Reference

## Headings

```
# Heading level 1
## Heading level 2
...
###### Heading level 6
```

## Paragraphs

An empty line marks the end of a paragraph.

```
Paragraph 1 ...
More text in paragraph 1.

Paragraph 2 ...
```

Paragraph 1 ... More text in paragraph 1.

Paragraph 2 ...

## Italics

```
_This is in italics_

This *is also italics* but can't start a line with * since it will be treated as list.
```

*This is in italics*

This *is also italics* but can't start a line with * since it will be treated as list.

## Bold

```
__This text is bold.__

**This text is also bold.**
```

**This text is bold.**

**This text is also bold.**

## Blockquote

```
> This line is block quoted.
```

*This line is block quoted.*

## Lists

### Unordered lists (* or -)

```
* item 1
  Also part of item 1.

- item 2
    * item 2.1 (this is indented 4 spaces!)
```

• item 1 Also part of item 1.
• item 2• item 2.1 (this is indented 4 spaces!)

## Ordered list (n.)

```
1. item 1.
2. item 2.
    1. item 2.1
```

1. item 1.
2. item 2.2.1. item 2.1

Please note that the actual numbers does not matter! They could all be "1."! The items will be enumerated automatically in order no matter what numbers you enter in the source.

## *Code block (pre formatted with a fixed width font)*

Each line starting with a tab or 4 spaces are considered belonging to a pre formatted block.

```
This
    is
        a
            preformatted
                block!
```

## *Horizontal rule*

Any of:

```
* * *
***
********...***
- - -
----
----------...---
```

## *Links*

```
[This is a link to markdown syntax on
    daringfireball.net](http://daringfireball.net/projects/markdown/syntax)
```

This is a link to markdown syntax on daringfireball.net

Short "autolink" version:

```
<http://www.daringfireball.net>
```

http://www.daringfireball.net

## *Images*

```
![Alt text](/path/to/img.png)

![Alt text](/path/to/img.png "title")
```

Note that for **MarkdownDoc** "/path/to/img.png" can be one of the following:

• A http URL.

- A local path which is resolved by first looking at the current directory. If not found the parent directory is tried. This repeats until root directory is reached. You should never use ".." in these paths!
- A local path relative to the *.md* document referencing it. In this case "../images/myimg.png" is valid.

"file:" URLs are also allowed and behave the same as local paths.

## *backslash (\)*

The \ character can be used to escape characters that have markdown meaning. \\ will for example produce \. * will produce *.

# Licenses

## *Project License*

**Apache Software License version 2.0**

## *Third Party Licenses*

**Apache Software License version 2.0**

The following third party products are using this license:

- OptionsManager-2.0.3
- annotations-13.0
- pdfbox-2.0.2
- groovy-all-2.4.4

# License Texts