

# Madoko Reference

## A Fast Scholarly Markdown Processor

2016-12-12 (version 1.0.3)

Daan Leijen  
Microsoft Research  
daan@microsoft.com

<b>1. Introduction</b>	5
1.1. Madoko philosophy	5
1.2. License	7
<b>2. Installation and usage</b>	9
<b>3. Syntax: Inline elements</b>	11
3.1. Emphasis	11
3.2. Code	11
3.3. Sub- and super-script	12
3.4. Strike-out	12
3.5. Smart quotes, symbols, and direct links	12
3.5.1. Advanced: changing quotes	13
3.6. Links	14
3.7. Images	14
3.7.1. Image formats	15
3.8. Footnotes	15
3.9. Escape sequences	16
3.9.1. Special escapes	16
<b>4. Syntax: Block elements</b>	19
4.1. Headings and rules	19
4.2. Identities and labels	20
4.2.1. A named heading	20
An unnumbered heading	21
A labeled heading	21
4.3. Figures and Table Figures	21
4.3.1. Advanced: sub-figures	22
4.4. Lists	23
4.5. Definition lists	25
4.6. Block quotes	27
4.7. Columns: putting blocks next to each other	27
4.8. Code blocks	28
4.8.1. Syntax highlighting	29
4.8.2. Escaped code	30
4.8.3. Advanced: Pretty code alignment	32
4.8.4. Advanced: Customizing highlight colors	34
4.8.5. Advanced: Custom syntax highlighting	34
4.8.6. Advanced: taking over code blocks	36
4.9. Tables	36
4.9.1. Horizontal rules	38
4.9.2. Long tables	39
4.9.3. The width of columns	39
4.9.4. Colors	39
4.9.5. Book tables	40
4.10. Mathematics	41
4.10.1. Plain math and alignment	42
4.10.2. Using packages	43
4.10.3. Math commands	44

4.10.4. Mathematics in HTML . . . . .	45
4.10.5. Advanced: Generic LaTeX snippets . . . . .	45
4.10.6. Advanced: Efficiency of math rendering . . . . .	50
4.10.7. Advanced: Preformatted math . . . . .	50
4.10.8. Setting all code to preformatted math . . . . .	52
4.11. Table of contents . . . . .	52
4.11.1. Advanced: Custom tables of contents . . . . .	53
4.12. Bibliography and Citations . . . . .	54
4.12.1. Bibliography styles . . . . .	56
4.12.2. Citation styles . . . . .	57
4.12.3. Bibliography tooltips and searches . . . . .	58
4.13. Custom blocks . . . . .	59
4.13.1. Predefined custom blocks . . . . .	60
4.14. Special block elements . . . . .	62
4.14.1. Advanced: including file fragments . . . . .	63
<b>5. Syntax: Metadata and Styling</b> . . . . .	<b>65</b>
5.1. Metadata . . . . .	65
5.1.1. Special metadata keys . . . . .	65
5.1.2. HTML keys . . . . .	68
5.1.3. LaTeX keys . . . . .	71
5.1.4. Conditional metadata . . . . .	73
5.2. Entities . . . . .	73
5.3. CSS Attributes and Styling . . . . .	75
5.3.1. CSS Attributes . . . . .	75
5.3.2. CSS formatting support . . . . .	76
5.3.3. CSS font family . . . . .	78
5.3.4. CSS colors . . . . .	79
5.3.5. Complex CSS Layout . . . . .	79
5.3.6. Floating blocks . . . . .	82
5.3.7. Special attributes . . . . .	83
5.3.8. Special attribute classes . . . . .	84
5.4. Metadata rules . . . . .	85
5.4.1. Names of predefined elements . . . . .	86
5.4.2. Advanced: Styling in CSS . . . . .	87
5.5. Numbering . . . . .	87
5.5.1. Advanced: Custom numbering . . . . .	87
5.5.2. Advanced: Reset counters . . . . .	88
5.5.3. Advanced: Display format . . . . .	88
5.6. Advanced: Replacement . . . . .	89
5.6.1. Advanced: Regular expression replacement . . . . .	90
5.6.2. Advanced recursion and replacement . . . . .	91
<b>References</b> . . . . .	<b>93</b>
<b>A. Appendix</b> . . . . .	<b>95</b>
A.1. Command line options . . . . .	95
A.2. Slide shows and presentations . . . . .	96
A.2.1. Using Reveal.js . . . . .	97

A.2.2. Using Beamer . . . . .	98
A.3. Advanced: Customizing citations . . . . .	98
A.4. Advanced: Not using BibTeX . . . . .	99
A.5. Advanced: packages in dynamic math mode . . . . .	100
A.6. Advanced: Using Prettify to highlight code . . . . .	101
A.7. Advanced styling in LaTeX . . . . .	101
A.8. Unicode characters . . . . .	102
A.8.1. Unicode in LaTeX . . . . .	103
A.8.2. Unicode font selection in LaTeX . . . . .	103
A.9. Recognized character entities . . . . .	104
A.10. Definitions of predefined custom blocks . . . . .	113
A.11. License and attribution . . . . .	114

# Chapter 1

## Introduction

Madoko is a fast javascript [Markdown](#) processor written in [Koka](#). It started out as a demo program for the new, strongly typed, [Koka](#) language and the name comes from “*Markdown in Koka*”.

Madoko can both be run local as a command-line program, or as a full on-line experience on [Madoko.net](#) with storage and collaboration through [dropbox](#) or [github](#).

The online editor can also edit files on the local disk, or run LaTeX locally, using the [madoko local](#) program.

### 1.1. Madoko philosophy

The main design goal of Madoko is to enable light-weight creation of high-quality scholarly<sup>1</sup> and industrial documents for the web and print, while maintaining John Gruber’s Markdown philosophy of simplicity and focus on plain text readability.

The popularity of Markdown is not accidental, and it is great for writing prose: it is super simple and straightforward to create good looking HTML documents. But for more serious use Markdown falls short in several areas, and Madoko provides many essential additions for larger documents:

- Extensive support for labeling and in-document [references](#).
- [Tables](#) with custom borders, column alignment, multicolumn spans, colors etc.
- Great [citation](#) support, using standard BibTeX entries with either BibTeX styles (`.bst`) or Citation Language styles (`.csl`).
- Excellent support for [advanced mathematics](#) ( $e^{i\pi}$ ) (and powerful packages like TikZ and PSTricks) with scalable vector images (SVG) in web pages.
- Great  $\text{\LaTeX}$  integration where one can use any LaTeX style or package, and any LaTeX document styles, like the ones provided by most publishers.

---

<sup>1</sup>There is actually an effort to define [scholarly markdown](#).

- Excellent support for static [syntax highlighting](#) and transformation of code fragments.
- Automatic [numbering](#) of sections, figures, examples, etc.
- Title page and [table-of-contents](#) generation.
- Support for user defined [custom blocks](#), like **Theorem**, **Abstract**, **Figure**, etc.
- Styling for both HTML and LaTeX output through [standard CSS attributes](#) and [CSS metadata rules](#).
- Powerful document transformations through [replacement rules](#).
- Create great [presentations](#) for both the web and in print.
- Compatible with most common Markdown extensions, such as [footnotes](#), [mathematics](#), [attributes](#), etc.

Moreover the online editor [Madoko.net](#) supports:

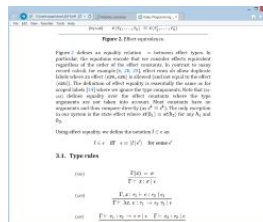
- Full online editing, previewing, and document generation.
- Seamless sharing and collaboration with other authors through [dropbox](#) and [github](#); it is very easy to work together on a document where changes are continuously merged.
- Full support for mathematics and bibliographies through server-side LaTeX; no need to install LaTeX yourself.
- This is an HTML5 app and also works offline. With the [madoko local](#) program you even access local files on disk, and run LaTeX locally.

Instead of a plethora of backends, Madoko concentrates on generating either HTML or high-quality PDF files through [LaTeX](#). There has been a lot of effort in Madoko to make the LaTeX generation robust and customizable while integrating well with the various academic document- and bibliography styles. This makes it great for writing articles using just Madoko and get both a high-quality print format (PDF) and a great looking HTML page. Look at this scientific article for an example:

LaTeX/PDF



HTML

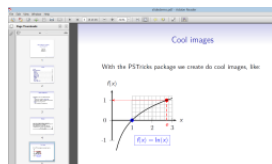


Edit in Madoko.net

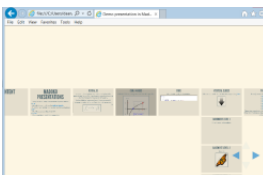


or this slide-show presentation:

## LaTeX/PDF



## HTML



## Edit in Madoko.net

```

1 [MADOKA presentation.mh]
2 Title : Some presentation in Madoko
3 Sub Title : In both HTML and PDF
4 Author : Dan LaZar
5
6 Affiliation : Microsoft Research
7 Email : danlazar@microsoft.com
8
9 Name: Name: Dan LaZar
10 Name: Name: Singapore
11 Package : pdfplots
12 Math Level : 0-9
13 @html ,Math-inline ,Math-Display: color=red;
14
15 [TEXT]
16
17 # Content
18 [THE]
19
20 # Madoko Presentations
21
22 Using Madoko it is easy to create beautiful presentations.
23
24

```

Also, you can look at the [PDF](#) and [source](#) of this document. In the future, we plan to support e-books through the [ePub](#) format too.

Here are some other (large!) examples of complicated documents, click on the links to edit the document directly in [Madoko.net](#):

- [The anatomy of programming languages](#): A large part of the [AoPL](#) book by Prof. Dr. William Cook.
- [Software model checking with IC3](#): A math-heavy presentation for the VTSA summer school 2014 by Dr. Nikolaj Bjørner and others.
- [The Madoko reference manual](#): This document by yours truly.

## 1.2. License

Madoko is a javascript program (written in [Koka](#)) that runs on [Node.js](#). Madoko is free software available under the [Apache 2.0 license](#) at [madoko.codeplex.com](#). Madoko uses various other open-source libraries, see [Appendix A.11](#) for a full list.





## Chapter 2

# Installation and usage

The recommended way to use Madoko now is online on [Madoko.net](https://madoko.net) in combination with [Dropbox](#) or [Github](#) to store your files and collaborate and share with other users.

Nevertheless, Madoko works great as a command line tool as well, and it is good to have a solid backup. The easiest way to use Madoko is as a command line tool is by using Node.js (which works on many platforms, like Windows, MacOSX, Linux etc). Madoko can also run inside a web browser or as a .NET executable. Installation under Node.js is very easy:

- Ensure you have [Node.js](#) installed on your system (and ensure that the Node installation directory is in your `PATH`).
- Open a command line window and run the Node package manager to install Madoko:

```
npm install madoko -g
```

and you are done. Translating a markdown document is done simply as:

- `madoko -v mydoc.mdk`

which generates `mydoc.html`. The `-v` flag gives more verbose output. To also generate a PDF file, use:

- `madoko --pdf -vv --odir=out mydoc`

where `--odir` puts all output files in the `out` directory. To generate a PDF, you need to have LaTeX installed on your system, which is also required for mathematics and bibliographies. We recommend the full [TeXLive](#) LaTeX system as it is available for Windows, Linux and MacOSX, and is used on the [Madoko.net](https://madoko.net) server as well.

For PDF output, we added an extra verbose flag in order to see any warnings LaTeX produces. A full description of all command line options can be found in Appendix [A.1](#).



## Chapter 3

# Syntax: Inline elements

Madoko is fully compatible with basic [Markdown syntax](#) and passes the entire test suite. It also implements most extensions, like [Github flavored markdown](#), [PanDoc](#), [Markdown Extra](#), and [multi-markdown](#), and it adds quite a few features itself to make it really useful for writing academic and industrial documents.

We assume that the reader is familiar with [basic markdown syntax](#). In Madoko, *tab*'s are considered to be equivalent to 4 spaces. It is therefore best to configure your editor to view tabs as 4 spaces wide or documents may look off.

### 3.1. Emphasis

Enclose words in asterisks (\*) or underscores (\_) to emphasize them. Use double asterisks or underscores for strong emphasis:

```
Here is _some emphasis_, or using *asterisks*.  
Or use __strong emphasis__, or like **this**.
```

---

Here is *some emphasis*, or using *asterisks*. Or use **strong emphasis**, or like **this**.

### 3.2. Code

You can include pre-formatted text in regular text using back-quotes (`).

For emphasis, use the `<strong>` tag in HTML.  
 We can use back-quotes by using multiple ``back`quotes``.

---

For emphasis, use the `<strong>` tag in HTML. We can use back-quotes by using multiple `back`quotes`.

Section 4.8 discusses code in more detail including syntax highlighting.

### 3.3. Sub- and super-script

Using tilde (~) and hat (^), you can format inline text as sub- and super-script respectively. Inside script, no white space is allowed to prevent mistakes. If you need white space you can still use an escaped space (\ ).

Here is how you write H-2~0 or E=MC^2~.  
 Please use escapes for~longer\ script~.

---

Here is how you write H<sub>2</sub>O or E=MC<sup>2</sup>. Please use escapes for~longer script~.

### 3.4. Strike-out

Enclose anything in two tildes and it will strike out the content:

There is a ~~strike out~~ here.

---

There is a ~~strike out~~ here.

### 3.5. Smart quotes, symbols, and direct links

Madoko will quote smartly using proper open and closing quotes for anything enclosed in single quotes ('), double quotes ("), or French quotes (<< and >>).

"double quoted"  
 'single quoted'  
 <<guillemot quoted>>

```
<http://www.google.com>
```

---

```
“double quoted”
```

```
’single quoted’
```

```
«guillemot quoted»
```

```
http://www.google.com
```

Note that text enclosed in < and > brackets gets interpreted as direct link. Madoko will only smart quote single quotes if the last quote is not directly followed by a letter. This often prevents wrong quotation with words like “can’t”:

```
'really, I can't do this', he said.
```

```
I can't and mustn't do this.
```

---

```
‘really, I can’t do this’, he said. I can’t and mustn’t do this.
```

Madoko will also replace multiple dashes and dots to a proper symbol:

```
Please distinguish a minus sign, -, from the _en dash_ which
is used to separate spans or pages, like 1--20, and the
_em dash_ which is even longer and sometimes used for
quotation attribution. --- Oscar Wilde.
Three dots ... should be close together.
```

---

```
Please distinguish a minus sign, -, from the en dash which is used to sep-
arate spans or pages, like 1–20, and the em dash which is even longer and
sometimes used for quotation attribution. — Oscar Wilde.
Three dots ... should be close together.
```

### 3.5.1. Advanced: changing quotes

When replacing quotes, Madoko just inserts the right entity, like an `&ldquo;`; for a left double quote (“). As described in Section 5.2 we can redefine entities, and use that to change the quotes. For example, in some obscure countries, like the Netherlands, people like to start a quotation at „the bottom” instead. You can get this behavior by adding the following `metadata` rule:

```
ldquo: &bdquo;
```

Or Japanese style:

```
lsquo: [⌘12300;]{font-family:"MS Gothic"}
rsquo: [⌘12301;]{font-family:"MS Gothic"}
ldquo: ⌘12302;
rdquo: ⌘12303;
```

to single quote 「like this」

### 3.6. Links

Madoko has three kinds of links, reference links, inline links, and direct links. The inline links have the linked text between square brackets and the URL follows between parenthesis, while direct links are simply enclosed between angled brackets:

```
Here is a link to [Google](http://www.google.com).
Or as a direct link: <http://www.google.com>.
```

---

Here is a link to [Google](http://www.google.com). Or as a direct link: <http://www.google.com>.

Generally, reference links are preferred. Here, the link is defined separately after the body of text such that it looks less cluttered:

```
Here is a link to [Google] again.
We can also [Change the text][Google].

[Google]: http://www.google.com "Google"
```

---

Here is a link to [Google](http://www.google.com) again. We can also [Change the text](#).

Note how [Google] is simply a shorthand for [Google] [Google]. The title in double quotes in the link definition is optional. Link definitions do not have to follow the text immediately and can be defined anywhere in the document.

### 3.7. Images

Images are included using a regular link prefixed with an exclamation mark (!).

```
A butterfly: ![bfly].
```

```
[bfly]: images/butterfly-200.png "A Monarch" { width: 100px }
```



This example also shows the use of attributes (Section 5.3.1) where we can specify the `width`, `height`, `vertical-align`, `zoom`, and `transform-rotate` attributes of the image.

**Madoko.net:** You can use the **Insert File** menu to include images, or just drag&drop them directly into the editor. Just watch the file size as images larger than about 1Mb are rejected by the Madoko server.

### 3.7.1. Image formats

Sometimes different backends require different image formats. Often for PDF output, a `.pdf` or `.eps` image is preferred while this format is not supported in the HTML backend. It is possible to provide images in multiple formats – for convenience, the HTML backend will by default try to load a `.png` image if a `.pdf` or `.eps` image is specified.

More generally, you can provide selection pattern as a comma separated list in the image url, for example:

```
images/butterfly-200.[ps,svg,png]
```

This specifies three available images where each backend chooses one based on the best results for that backend. The default extensions that are supported by each backend in order of preference are:

- html: `.svg`, `.png`, `.jpg`, `.jpeg`, `.gif`, `.tif`, `.tiff`, `.bmp`, `.jpx`, `.jp2`.
- pdf (latex): `.pdf`, `.eps`, `.ps`, `.png`, `.jpg`, `.jpeg`.

## 3.8. Footnotes

Footnotes are written as regular link definitions prefixed with a hat (^) character. This is the syntax [originally proposed](#) by John Gruber.

```
Here is a footnote[^fn].
```

```
[^fn]: This is the content of the example footnote.
      You can continue a footnote by indenting content.
      And notice the back link.
```

Here is a footnote<sup>2</sup>.

### 3.9. Escape sequences

If you want to use a special character directly without a special Markdown meaning, precede it with a backslash (\). For example, to use a star (\*) without causing emphasis, you can write \\*.

Madoko will never escape a letter or digit and always keep the backslash, so \a becomes “\a” while \& becomes just “&”. Thus, you can safely write windows style file names without needing an escape:

Would you like c:\foo\bar to be deleted? Yes\No.  
Here are some other escapes: \\, \#, \\*, \|, and \0.

Would you like c:\foo\bar to be deleted? Yes\No. Here are some other escapes: \, #, \*, |, and \0.

This approach is different than that of Markdown which only escapes a specific set of characters, while Madoko escapes everything that is not a letter or digit. The advantage of the approach of Madoko is that this is easy to remember, while trying to remember a specific set of special escape characters is near impossible. This is similar to the [PanDoc](#) approach to escape sequences.

#### 3.9.1. Special escapes

Some characters are translated specially when escaped. If you escape a space (\ ), it is translated as a non-breakable space, while a backslash at the end of a line causes a hard line break to be inserted. The latter is recommended over using the standard Markdown way of using two spaces at the end of a line because it leaves visual clue that a line break occurs.

Here is non\ breakable space and a hard\  
line break with a \\* star.

Here is non breakable space and a hard  
line break with a \* star.

<sup>2</sup>This is the content of the example footnote. You can continue a footnote by indenting content. And notice the back link.



---

Finally, the escape sequence `\/` translates to nothing; this can be very useful to separate certain constructs. For example, emphasis is suppressed if the underscores appear inside a word, as in `my_example_here`. Using the empty escape sequence we can emphasize inside words too: to get `myexamplehere`, we can simply write `my\_example\_here`.



## Chapter 4

# Syntax: Block elements

### 4.1. Headings and rules

Headings are written by prefixing with one or more hash characters (#):

```
# A level one heading
## A level two heading
### A level three heading
...
##### Up to level six
```

It is also possible to write level one and level two headers using a line of three or more equal (=) or dash (-) characters:

```
A level one heading
=====
```

```
A level two heading
-----
```

But hash-headings are generally preferred since a three or more dashes (or underscores or asterisks) are also used for horizontal rules:

Above...

-----

and below the line.

\_\_\_\_\_

Above...

\_\_\_\_\_

and below the line.

## 4.2. Identities and labels

Madoko has extensive support for numbering and referencing elements in a document. Elements are given an identity using attributes (see Section 5.3.1). For example, here we give an identifier `myheading` to a heading:

```
### A named heading { #myheading }
```

And we can refer to it

- \* Using an explicit `[link](#myheading)`  
(or `[reference][#myheading]`).
- \* Or using an implicit link to Section `[#myheading]`.
- \* Or we can just see its label, namely `&myheading;`.

### 4.2.1. A named heading

And we can refer to it

- Using an explicit `link` (or `reference`).
- Or using an implicit link to Section 4.2.1.
- Or we can just see its label, namely 4.2.1.

Using an implicit link is generally recommended. When type setting a reference such as Section 4.2.1, Madoko automatically inserts a non-breakable space between “Section” and the reference<sup>3</sup>. Unlike LaTeX there is no need to insert such non-breakable space yourself.

Of course, we can refer to *any* element that has an identity, like equations, figures, tables, etc. Here is an example with an equation (see Section 4.10):

In Equation `[#euler]` we define `[Euler]`'s number  $e$ :

```
~ Equation { #euler; caption:"Euler's formula" }
e = \lim_{n\to\infty} \left( 1 + \frac{1}{n} \right)^n
~
```

`[Euler]`: [http://en.wikipedia.org/wiki/Euler's\\_number](http://en.wikipedia.org/wiki/Euler's_number)

<sup>3</sup>Madoko will still use a regular space if you use an explicit line break between “Section” and the reference in the source.

In Equation (1) we define Euler’s number  $e$ :

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n \quad (1)$$

An implicit link such as `[#myheading]` is a shorthand for an explicit link of the form `[&myheading;](#myheading)`, i.e. a local reference to `#myheading` which displays the entity `&myheading;`. In Madoko, an entity name like `&myheading;` is replaced by the *label* value of the referred block (see Section 5.2). For headers, this is by default the header number (as we will see in Section 5.5) but you can set any label you’d like. Using a minus (-) in the attributes clears all attributes, which is used in the following example to suppress the default numbering:

```
### An unnumbered heading { - }
### A labeled heading      { -; id: myheading1; label: "my label" }
Let's refer to Section [#myheading1].
```

---

**An unnumbered heading**

**A labeled heading**

Let’s refer to Section [my label](#).

Also, any local link, like “Section 4.2.1” or a citation like “[2]”, displays a tooltip when hovering above it. The tooltip content is determined by the `caption` attribute of the target element. For headings, this is by default the heading text.

**Madoko.net:** The editor will automatically display and complete section references (and citations) once you start typing a # (or @). You can also press **Ctrl+Space** to invoke auto-completion explicitly.

## 4.3. Figures and Table Figures

Figures can be included using the **Figure** custom block (see Section 4.13):

```
Figure [#fig-monarch] shows how to put an image in a figure.

~ Figure { #fig-monarch; caption: "A Monarch butterfly." }
The ![monarch] image.
~
```



**Figure 1.** A Monarch butterfly.

```
[monarch]: butterfly-200.png { width:100px; vertical-align:middle }
```

Figure 1 shows how to put an image in a figure.

A figure can be given an identity and referred to just like headings. The `caption` attribute gives the caption of the figure which is also used in the table-of-figures. A `Figure` can have a `page-align` attribute that can be set to `top`, `bottom`, `topbottom`, `page`, `here`, `forcehere`, and `inplace`. This is ignored in the HTML output but used in LaTeX to influence where a figure is placed on a page. The `.wide` attribute is used in LaTeX to have figures span both columns in a two-column document class.

Besides regular figures, there is also the `Table Figure` element for tables:

Table `[#tab-sample]` shows an example table figure.

```
~ TableFigure { #tab-sample; \
    caption: "Modelle mit unterschiedlich geschätztem baseline hazard"; }
|           | $c(t)$           | |||
|           |-----|-----|-----|-----|
|           | (A0)   | (A1)   | (A2)   | (A3)   |
|           | ohne   | Log     | Polynom | Stückweise konstant |
|:-----: |:-----: |:-----: |:-----: |:-----:
| Log likelihood | -6.798 | -6.733 | -6.715 | -6.686 |
| Pseudo  $R^2$  | 0,048 | 0,057 | 0,059 | - |
| AIC          | 13.615 | 13.489 | 13.456 | 13.483 |
| BIC          | 13.711 | 13.594 | 13.580 | 14.009 |
| N            | 105.484 | 105.484 | 105.484 | 105.484 |
|-----|-----|-----|-----|-----|
{ .booktable }
~
```

Table 1 shows an example table figure.

### 4.3.1. Advanced: sub-figures

You can also place multiple sub-figures inside a figure using the `SubFigure` and `SubFigureRow` blocks. Here is an example:

	$c(t)$			
	(A0)	(A1)	(A2)	(A3)
	ohne	Log	Polynom	Stückweise konstant
Log likelihood	-6.798	-6.733	-6.715	-6.686
Pseudo $R^2$	0,048	0,057	0,059	-
AIC	13.615	13.489	13.456	13.483
BIC	13.711	13.594	13.580	14.009
N	105.484	105.484	105.484	105.484

**Table 1.** Modelle mit unterschiedlich geschätztem baseline hazard

```

~ Begin Figure { #fig-subfigs; caption:"Examples of sub figures." }
~ Begin SubFigureRow { vertical-align: bottom }
~ SubFigure { #fig-subfig1; caption:"A butterfly" }
! [monarch]
~
~ SubFigure { #fig-subfig2; caption:"A matrix"; }
~~ Math
A_{m,n} =
\begin{pmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{pmatrix}
~~
~
~ End SubFigureRow
~ Begin SubFigureRow
~ SubFigure { #fig-subfig3; caption:"A textual subfigure" }
One more sub-figure with some text.
~
~ End SubFigureRow
~ End Figure

We can refer to the Monarch sub-figure [#fig-subfig1] just like any other figure.

We can refer to the Monarch sub-figure 2a just like any other figure.

```

## 4.4. Lists

Unordered lists are created simply by using an asterisks (\*), plus (+), or dash (-) as item markers preceded by an empty line (or otherwise it is considered



(a) A butterfly

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

(b) A matrix

One more sub-figure with some text.

(c) A textual subfigure

---

**Figure 2.** Examples of sub figures.

part of a paragraph):

Groceries:

- \* Banana
- \* Bread
  - white
  - whole grain
- \* Basil

---

Groceries:

- Banana
- Bread
  - white
  - whole grain
- Basil

Each item marker must be followed by a space. You can created ordered lists using numbers followed by a dot and space. Also long list items can be wrapped by indenting the text for each item:

1. An ordered list example.  
With some longer items.
2. An another item  
with more text.

- 
1. An ordered list example. With some longer items.



2. An another item with more text.

Normally, the content of each list item is just treated as text and not as a paragraph. If there are any blank lines in the entire list, each item is typeset with a surrounding paragraph which makes the list a bit looser for HTML output:

3. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.

3. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.

3. Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

---

3. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.

4. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.

5. Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

Note how the specific numbers used in the list do not matter, except that the first item determines the start number of the enumeration.

For ease of formatting, unordered lists get a class assigned corresponding to their first bullet:

1. A bullet `*` gets class `list-star` (default display as `•`).
2. A bullet `+` gets class `list-plus` (default display as `■`).
3. A bullet `-` gets class `list-dash` (default display as `-`).

This can be used in CSS or Latex to customize how the bullets are displayed.

## 4.5. Definition lists

Definition lists are used to define terms. Each term must fit on one line and is followed by one or more definitions. Each definition starts with one or two spaces followed by a tiled (`~`) or colon (`:`). Each definition must be indented by four spaces.

```

Abstract syntax
~ The conceptual structure (illustrated by the pictures) is called
  the abstract syntax of the language.
Concrete syntax
~ The particular details and rules for writing expressions as strings
  of characters is called the concrete syntax.
~ Perhaps some other meaning too?

```

---

#### **Abstract syntax**

The conceptual structure (illustrated by the pictures) is called the abstract syntax of the language.

#### **Concrete syntax**

The particular details and rules for writing expressions as strings of characters is called the concrete syntax.  
Perhaps some other meaning too?

Madoko has an extension to also allow definition lists to be specified where each term started with as a regular `*` list and each definition with a colon `::`

```

* Abstract syntax
: The conceptual structure (illustrated by the pictures) is called
  the abstract syntax of the language.
* Concrete syntax
: The particular details and rules for writing expressions as strings
  of characters is called the concrete syntax.

```

---

#### **Abstract syntax**

The conceptual structure (illustrated by the pictures) is called the abstract syntax of the language.

#### **Concrete syntax**

The particular details and rules for writing expressions as strings of characters is called the concrete syntax.

The advantage of using this style of definition lists is that it makes your document more portable: when a simpler markdown processor is used, such list will still render somewhat nicely as:

- Abstract syntax : The conceptual structure (illustrated by the pictures) is called the abstract syntax of the language.
- Concrete syntax : The particular details and rules for writing expressions as strings of characters is called the concrete syntax.

## 4.6. Block quotes

Block quotes are written just as in email using > angle brackets:

Let's start a block quote:

```
> Of life's two chief prizes, beauty and truth,  
> I found the first in a loving heart and the  
> second in a laborer's hand.\n> &emsp;&emsp; --- Khalil Gibran
```

Let's start a block quote:

Of life's two chief prizes, beauty and truth, I found the first in  
a loving heart and the second in a laborer's hand.  
— Khalil Gibran

Just like lists, block quotes can be nested and contain other block elements.

#### 4.7. Columns: putting blocks next to each other

Using the `Columns` and `Column` custom blocks we can put block elements next to each other on a page. Use the `width` attribute to control the width of each column. Any columns without `width` are evenly divided over the remaining space.

[illegible]

```
A butterfly ![bfly]
~
~ Column { width:5em}
~~ Center
$e = mc^2$.
~~
~
~ End Columns
```

---

Here is a long paragraph. Here is a long paragraph. Here is a long paragraph. Here is a long paragraph. Here is a long paragraph. Here is a long paragraph. Here is a long paragraph. Here is a long paragraph.

A butterfly



$e = mc^2$ .

## 4.8. Code blocks

You can write preformatted code simply by indenting the code with a tab character or at least *four* spaces. For example:

```
<a href="foo.com">a link in html</a>
```

---

```
<a href="foo.com">a link in html</a>
```

Another way to write code is to use *fenced* code blocks. These start with at least three backticks (```) and goes on to the first line containing the same number of backticks. Moreover, you can include the language name after the backticks:

```
``` html
<a href="foo.com">a link in html</a>
```
```

---

```
<a href="foo.com">a link in html</a>
```

### 4.8.1. Syntax highlighting

As we can see from the previous example, Madoko will automatically perform (static) syntax highlighting for any code fragment that has the `language` key set. For a fenced code block, Madoko automatically adds the `language` key when the language is specified.

Internally, Madoko uses the Monarch library to do syntax highlighting. This means that also the PDF output through LaTeX will be fully highlighted (oh what a joy to no longer having to remember the vagaries of the `listing` package). The standard library contains some default language definition, like Java, JavaScript, C#, HTML, Ruby, and Python.

A Haskell keyword like `instance`<sup>{language:haskell}</sup> can be highlighted inline.

Or some Ruby code block:

```
``` Ruby
# ruby pi - how to calculate pi with ruby.
num = 4.0
pi = 0
plus = true
den = 1
while den < 10000000
  if plus
    pi = pi + num/den
    plus = false
  else
    pi = pi - num/den
    plus = true
  end
  den = den + 2
end

puts "PI = #{pi}" # calculated value of pi
```
```

---

A Haskell keyword like `instance` can be highlighted inline. Or some Ruby code block:

```
# ruby pi - how to calculate pi with ruby.
num = 4.0
pi = 0
plus = true
den = 1
while den < 10000000
  if plus
    pi = pi + num/den
```

```

    plus = false
  else
    pi = pi - num/den
    plus = true
  end
  den = den + 2
end

puts "PI = #{pi}"    # calculated value of pi

```

If you use a particular language a lot in your document, you may want to set it as the default language using a metadata rule. For example:

```

pre,code {
  language: JavaScript;
}

```

If you want to disable syntax highlighting for a particular code fragment, you can set the language key to empty. To disable highlighting completely, set the **Highlight** metadata key to **False**.

For HTML, it is also possible to use dynamic syntax highlighting with the [Prettify](#) library, see [Appendix A.6](#) for more information.

#### 4.8.2. Escaped code

Madoko allows you to escape back to inline Madoko syntax inside code by bracketing inside `\(` and `\)`. This can be nice to insert special looking symbols in the code while maintaining full automatic syntax highlighting for all the surrounding code:

```

``` javascript
function sqr( x ) {
  var \(&pi;\) = 3.141593;
  return x*x /* ensures \(_result_ $\ge$ 0\) on return.*/
}
```



---



function sqr( x ) {
  var  $\pi$  = 3.141593;
  return x*x /* ensures result  $\geq 0$  on return.*/
}

```

This is especially powerful in combination with replace attributes ([Section 5.6](#)).

For example, in this document, we have the following metadata rules:

```
pre,code {
  replace: "/==>/\(&rArr;\)/g";
}
```

which makes it easy to insert an implication symbol in code:

Look at implication (1) in the following code:

```
``javascript
function implies(x,y) { x ==> y } \((**1**)\)
``
```

---

Look at implication (1) in the following code:

```
function implies(x,y) { x  $\Rightarrow$  y } (1)
```

Each escaped piece of code is wrapped in a `code-escaped` element. This can be used to for example render all escaped text in a serif font:

```
.code-escaped {
  font-family: serif;
}
```

Finally, you can use the `.noescape` class to suppress escaping within a code block.

When encountering a replaced piece of text, the syntax highlighter treats it by default as whitespace. It is possible to specify though what characters the syntax highlighter sees using the bar format: `\(highlight|replacement\)` where *highlight* is used for the purpose of syntax highlighting (and alignment) and *replacement* is the replacement text in the output. For example, to colorize our implication symbol as a keyword, we can do:

Look at highlighted implication (1) in the following code:

```
``javascript
function implies(x,y) { x \(\return|\&rArr;\) y } \((**1**)\)
``
```

---

Look at highlighted implication (1) in the following code:

```
function implies(x,y) { x  $\Rightarrow$  y } (1)
```

### 4.8.3. Advanced: Pretty code alignment

When writing high quality articles containing code fragments, it often looks best when using a proportional font instead of monospace. However, it becomes hard to align the code properly in such cases. Madoko offers a special `pretty` mode for advanced code alignment inspired by the excellent [lhs2tex](#) tool by Andres Löh.

We use a proportional font, but everything preceded by 2 or more spaces is aligned properly:

```
``` Haskell { .pretty }
data Exp  = Number    Int
          | Add        Exp Exp
          | Subtract   Exp Exp
          | Multiply    Exp Exp
          | Divide      Exp Exp
          | Variable    String      -- added
          deriving (Eq)
```
```

We use a proportional font, but everything preceded by 2 or more spaces is aligned properly:

```
data Exp = Number Int
        | Add    Exp Exp
        | Subtract Exp Exp
        | Multiply Exp Exp
        | Divide   Exp Exp
        | Variable String -- added
        deriving (Eq)
```

Even though the font used is proportional everything which is preceded by 2 or more spaces is aligned. If something is indented more, there is some default indentation being added. For example:

```
``` haskell { .pretty; replace: "/->/\(->|&rarr;\)/g"; }
substitutel :: (String, Int) -> Exp -> Exp
substitutel (var, val) exp = subst exp where
  subst (Number i)      = Number i
  subst (Add a b)        = Add (subst a) (subst b)
  subst (Subtract a b)   = Subtract (subst a) (subst b)
  subst (Multiply a b)   = Multiply (subst a) (subst b)
  subst (Divide a b)     = Divide (subst a) (subst b)
```



```

subst (Variable name) = if var == name
                        then Number val
                        else Variable name
...

```

---

```

substitute1 :: (String, Int) → Exp → Exp
substitute1 (var, val) exp = subst exp where
  subst (Number i)      = Number i
  subst (Add a b)       = Add (subst a) (subst b)
  subst (Subtract a b)  = Subtract (subst a) (subst b)
  subst (Multiply a b)  = Multiply (subst a) (subst b)
  subst (Divide a b)    = Divide (subst a) (subst b)
  subst (Variable name) = if var == name
                        then Number val
                        else Variable name

```

Note that we used a replacement expression to replace arrows with their proper symbol. To make alignment and syntax highlighting work properly, we used the bar format. In this case we wrote `\(->|` which makes the replacement count for 2 characters for alignment purposes (and uses `->` for the syntax highlighter). In LaTeX the samples generally align very nice.

Finally, for really pretty code, it generally helps to add more replacements. For example, here is a sample from the [lhs2tex](#) manual:

```

... Haskell { .pretty; \
    replace:"//->/\(->| [&rarr;]{.serif}\)/\\(?![()])/\(&lambda;\)/g"; \
    replace:"/_(?:=[a-zA-Z])/(\_|&lowline;)/g"; \
    replace:"/([a-zA-Z])(\d)\b/(\1~\2~)/g"; \
  }
rep_alg      = ( \_      -> \m -> Leaf m
                , \lfun rfun -> \m -> let lt = lfun m
                                      rt = rfun m
                                      in Bin lt rt "hi"
                )
replace_min t = (cata_tree rep_alg t) (cata_tree min_alg t)
...

```

```

rep_alg      = ( λ_      → λm → Leaf m
                , λlfun rfun → λm → let lt = lfun m
                                   rt = rfun m
                                   in Bin lt rt "hi"
                )
replace_min t = (cata_tree rep_alg t) (cata_tree min_alg t)

```

Note that to improve the rendering, we added replacements for underscores inside identifiers with `&lowline`; and subscripted digits that end an identifier. Also, we replaced right arrows and lambda bindings. See Section 5.6.2 for more information about complex replacements.

#### 4.8.4. Advanced: Customizing highlight colors

The syntax highlighter for a language assigns specific class names to each token. Using these classes we can customize how they are displayed. The standard class names that are often used include:

identifier, operators, keyword, string, escape, comment, commentdoc, constant, entity, tag, info, warn, error, debug, regexp, attribute, value, constructor, namespace, header, type, predefined, invalid, code, codekeyword, typevar, delimiter, number, variable, meta, bold, and italic.

To customize the appearance we can just use a CSS rule in the metadata section where we combine a token class name with `.token`:

```

.token.identifier { font-style: italic }
.token.string.escape { color: gray }

```

For identifiers in *pretty* code (Section 4.8.3), the token rendering is determined in combination with the `.ptoken` class:

```

@if tex {
  .ptoken.keyword { font-family: sans-serif }
}

```

#### 4.8.5. Advanced: Custom syntax highlighting

Madoko can actually load syntax specifications dynamically to perform syntax highlighting. For example, suppose you are a blogger that is writing about the new Javascript module proposal, and you would like to highlight the new `module` or `export` keywords. Unfortunately, regular Javascript highlighting does no such thing:

```

// Javascript 5 modules
module Math {
  export function sum(x, y) {
    return x + y;
  }
}

```

```
export var pi = 3.141593;  
}
```

In Madoko, you can add your own syntax highlighter quite easily. First, register your new language definition in the metadata:

```
Colorizer: javascript  
Colorizer: javascript5
```

Because we are going to extend `javascript`, we also mention that language. Now you need a file `javascript5.json` which will contain the new language definition as JSON:

```
{ "name": "javascript5",  
  "extend": "javascript",  
  "extraKeywords": ["module", "export", "import"]  
}
```

This is generally all that is needed in most situations, but if you like to get fancy, there is extensive [documentation](#) about writing syntax extensions. Et voilà, with our new language definition we can properly highlight our previous example:

```
``` javascript5  
// Javascript 5 modules  
module Math {  
  export function sum(x, y) {  
    return x + y;  
  }  
  export var pi = 3.141593;  
}  
```\n\n-----\n\n// Javascript 5 modules  
module Math {  
  export function sum(x, y) {  
    return x + y;  
  }  
  export var pi = 3.141593;  
}
```

As an aside, note that colorization is done statically by Madoko, and not dynamically when a user views your document (and therefore, it works in LaTeX/PDF too).

There are many built-in languages in Madoko, and you can look at their JSON definitions. Here are a few: [Java](#), [Dafny](#), [C#](#), [Javascript](#), [HTML](#), [LaTeX](#),

[BibTeX](#), [Madoko](#), [Python](#), [Ruby](#), [SMT](#), [Koka](#), [C++](#) (named `cpp`), etc. If you write your own JSON description, you can simply include it yourself through the `Colorizer` metadata key.

#### 4.8.6. Advanced: taking over code blocks

It is possible to ‘take over’ code blocks and use them for example for mathematics. For example, here is how to make all code blocks use pre-formatted math (Section 4.10.7):

```
pre,code {
  input: mathpre;
}
```

Moreover, you can selectively only take over different kinds of code. Use `.coden` for inline code with  $n$  quotes, and `.pre-indented`, `pre-fenced`, or `pre-fencedn`, for indented or fenced code blocks (with  $n$  quotes). For example, here is how to use a default language only for indented and double quoted (``) code:

```
.pre-indented,.code2 {
  language: koka;
}
```

### 4.9. Tables

Madoko significantly extends the table syntax of basic Markdown. In particular, it is easy to add horizontal or vertical lines, to control cell alignment, use multiple column spans, colorize rows, etc. The basic rules for formatting a table are:

- Every line of the table should start and end with a `|` (or `+`) and columns are separated by `|` (or `+`) too. If you need a `|` character in cell content, use an escaped bar instead (`\|`).
- Every row can be on one line only, and there can be no blank lines.
- The table can optionally start with one or more *header* rows.
- A cell can span multiple columns by using multiple bars to end the cell, like `||` in the previous example.
- The table should always have a *column specifier* row that separates the header from the body of the table (or marks the start of the body in case there is no header). The content of each cell in the separator is just dashes (`-`) or tildes (`~`)

Here is an example of a plain table from “[Just a Theory](#)”.

id	name	description	price
:-----	:-----:	-----:	-----:
1	gizmo	Takes care of doohickies	1.99
2	doodad	Collects <i>*gizmos*</i>	23.80

10	dojigger		Foo		102.98	
1024	Self-explanatory, no?				0.99	

---

id	name	description	price
1	gizmo	Takes care of doohickies	1.99
2	doodad	Collects <i>gizmos</i>	23.80
10	dojigger	Foo	102.98
1024		Self-explanatory, no?	0.99

The column specifier row is the second row in the previous example, and it determines column alignment and vertical lines in a table:

- Columns can be aligned by using a : in a separator row column: one on the right or left aligns to the right or left, while a colon on both sides will center the column.
- If a column specifier uses a plus (+) instead of a bar (|) to separate the column, a vertical line is used to separate the columns. To distinguish the use of a + for a table instead of as a list item, there should be *no whitespace* following a + when used this way!
- If a column specifier cell uses dashes -, a horizontal line is drawn. By using tildes (~) instead, no line is drawn for that column specifier cell.

In the next example, we suppress the horizontal line after the header, but add some vertical lines. Also, we use two header rows.

	grouped					
id	name		description		price	
+:~~~~~	:~~~~~	:	~~~~~	+~~~~~	:+	
1	gizmo		Takes care of doohickies		1.99	
2	doodad		Collects <i>*gizmos*</i>		23.80	
10	dojigger		Foo		102.98	
1024	Self-explanatory, no?				0.99	

---

grouped				
id	name	description		price
1	gizmo	Takes care of doohickies		1.99
2	doodad	Collects <i>gizmos</i>		23.80
10	dojigger	Foo		102.98
1024		Self-explanatory, no?		0.99

[Madoko.net](http://Madoko.net): You can press **Alt-Q** to reformat tables (or paragraphs) and align all columns. Also, pressing **enter** in a table will add a new row (use **ctrl-enter** to use a regular line break if needed).

#### 4.9.1. Horizontal rules

We can draw horizontal rules in a table by using a row where every cell just contains dashes (-). By using equal signs (=) we get a double horizontal line. Here is a a table with no header and an outer border:

```
+ : --- : | - - - - - + : - - - - - : | - - - - : +
| centered gizmos || Takes care of doohickies | 1.99 |
| 2 | doodad | Collects *gizmos* | 23.80 |
| 10 | dojigger | Escaped \| and \+ | 102.98 |
| 1024 | thingamabob | Self-explanatory, no? | 0.99 |
| - - - - - | - - - - - | - - - - - | - - - - - |
```

---

centered gizmos	Takes care of doohickies	1.99
2 doodad	Collects <i>gizmos</i>	23.80
10 dojigger	Escaped   and +	102.98
1024 thingamabob	Self-explanatory, no?	0.99

And finally a complex table with all kinds of alignment, multiple column spans and horizontal rules – imagine trying to draw this example table in HTML or LaTeX without consulting the manual.

```
| ----- | ----- | ----- | ----- |
| id | name | description | price |
+-----+-----+-----+-----+
| 1 | gizmo | | |
| | ----- | | |
| 2 | doodad | Collect *gizmos* | 23.80 |
| ===== | ----- | ===== | ----- |
| 1024 | thingamabob | Self-explanatory | 0.99 |
| ----- | ----- | ----- | ----- |
```

---

id	name	description	price
1	gizmo		
2	doodad	Collect <i>gizmos</i>	23.80
1024	thingamabob	Self-explanatory	0.99

Note how we started the previous table with a horizontal line, where the column specifier row is on the third line.

### 4.9.2. Long tables

If a table is expected to be long and cross multiple pages, the table should be followed by an attribute declaration that sets the `breakable` attribute to `true`. This is used in LaTeX output to switch to a `longtable` environment which supports tables that can be broken over multiple pages.

### 4.9.3. The width of columns

We can specify the width (and other attributes) of a column by adding an `attribute` specification in the column specification row. In the following example, we use a column of fixed width, and one of a relative width with respect to the overall width of the table.

```
+:---:|---{width:2cm}---+:-----{width:60%}-----:+:---:|
| centered gizmos      || Take care of doo hickies | 1.99 |
| 2 | doodad          | Collects *gizmos* | 23.80 |
| 10 | dojigger        | Foo | 102.98 |
| 1024 | thingabob     | Self-explanatory, no? | 0.99 |
|-----|-----|-----|-----|
```

centered gizmos	Take care of doo hickies	1.99
2 doodad	Collects <i>gizmos</i>	23.80
10 dojigger	Foo	102.98
1024 thingabob	Self-explanatory, no?	0.99

### 4.9.4. Colors

Often we need to colorize certain rows or columns for clarity. Just like `width` we can specify a background color for a column in the column specifier row.

Moreover, we can specify after the table more attributes that can style the table further. In particular, any attribute can have the prefix `tr-` or `col-` to apply to rows or columns respectively. Furthermore, `tbody-tr-` and `thead-tr-` only apply to body or head rows. Moreover, you can follow the prefix with a modifier, `even-`, `odd-`, `last-`, or `n-`, to apply only to even, odd, the last, or the *n*th row or column. Finally, the prefixes `th-` and `td-` can be used for cell elements in the header or body. The following example shows this in action:

```

| ---- | ----- | ----- | ---- |
|  id |   name   | description | price |
+-----+---{background-color:Silver}---+-----+:|-----:++
| centered gizmos || Take care of doo hickies | 1.99 |
| 2 | doodad | Collects *gizmos* | 23.80 |
| 10 | dojigger | Foo | 102.98 |
| 1024 | thingabob | Self-explanatory, no? | 0.99 |
|-----|-----|-----|-----|
{ tbody-tr-odd-background-color:Gainsboro; \
  tr-even-background-color:Floralwhite }

```

id	name	description	price
centered gizmos		Take care of doo hickies	1.99
2	doodad	Collects <i>*gizmos*</i>	23.80
10	dojigger	Foo	102.98
1024	thingabob	Self-explanatory, no?	0.99

Colors are standard CSS colors and can be specified as described in Section 5.3.2.

#### 4.9.5. Book tables

For producing high quality tables, there are some general guidelines that need to be observed, in particular, it is considered good practice to:

1. Never use vertical rules in a table, and,
2. Never use double horizontal rules.

To make tables look good in this manner, we generally need rules of varying thickness to distinguish the top rule, from the mid- and bottom rules in a table. By using the `.booktable` class, the rule widths and rule separation is set automatically. Here is an example from the [Booktable](#) package documentation:

An example of a 'book table':

Item		
Animal	Description	&ensp;Price (\$)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33



```
| Armadillo&ensp; | frozen      | 8.99 |
| ----- | ----- | ----- |
{ .booktable }
```

An example of a ‘book table’:

Item		
Animal	Description	Price (\$)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Note how we used the empty escape sequence `\/` in front of the second dashed row in the header, or otherwise Madoko would interpret that row as the column specification row. The `.booktable` class ensures more vertical space between the rules, and a heavier top, mid-, and bottom rule. The exact styling is specified using a (built-in) [metadata rule](#) which is shown in [Appendix A.10](#).

## 4.10. Mathematics

Madoko documents can include mathematics in standard LaTeX syntax. Generally, inline math should be typeset between `$` characters, while block equations should use the `Equation` block syntax.

```
A famous equation is $E = mc^2$, but another famous one is:
~ Equation {#eq-gaussian; caption:"The Gaussian equation" }
\int_{-\infty}^{\infty} e^{-a x^2} d x = \sqrt{\frac{\pi}{a}}
~
and we can refer to Equation [#eq-gaussian] like any heading.
```

A famous equation is  $E = mc^2$ , but another famous one is:

$$\int_{-\infty}^{\infty} e^{-ax^2} dx = \sqrt{\frac{\pi}{a}} \quad (2)$$

and we can refer to Equation (2) like any heading.

Block equations can also be included use the `Math` custom block which is just like the standard LaTeX display environment (using `$$` or `\[`). However, the `Equation` block is preferred as it takes care of numbering, and alignment. See

also Section 4.13 on custom blocks that support mathematics, like **Theorem**, **Lemma**, **Proof**, etc. Madoko does not support the `\(`, `\[`, and `$$` commands to enclose mathematics since those are escape sequences in Madoko.

Sometimes inline equations between `$` signs can get quite long. In Madoko you can use newlines and put a long inline formula on multiple lines using LaTeX comments to end the line:

```
A long $E = %continue on next line
      mc^2$ formula.
```

#### 4.10.1. Plain math and alignment

Equations are generally numbered but sometimes we just want to display a mathematical equation without a number. The **Math** environment does just that:

```
~ Math
P\left(A=2\;;\middle|\frac{A^2}{B}>4\right)
~
```

---


$$P\left(A=2\;\middle|\;\frac{A^2}{B}>4\right)$$

Often, we also need to align formulas or display them on multiple lines. Instead of using the  $\text{\LaTeX}$  environments **align** and **gather**, we should use the environments **aligned** and **gathered** instead: the former are used in  $\text{\LaTeX}$  *outside* math mode and the latter are used when we are already in math mode (which is the case in Madoko). The **aligned** environment aligns formulas using an ampersand `&`:

```
~ Equation { #eq-alignment; caption:"Alignment example" }
\begin{aligned}
f(x) &= a x^2+b x +c & g(x) &= d x^3 \\
f'(x) &= 2 a x +b & g'(x) &= 3 d x^2
\end{aligned}
~
```

---


$$\begin{array}{ll} f(x) = ax^2 + bx + c & g(x) = dx^3 \\ f'(x) = 2ax + b & g'(x) = 3dx^2 \end{array} \quad (3)$$

Note that if you do this a lot, you can easily create our own `Aligned` custom block using the following metadata rule:

```
Aligned { replace: "~Math&nl;\begin{aligned}&nl;&source;&nl;\end{aligned}&nl;~" }
```

In that case, you can write simply:

```
~ Aligned
f(x) &= (x+a)(x+b) \\
&= x^2 + (a+b)x + ab
~
```

---


$$\begin{aligned} f(x) &= (x+a)(x+b) \\ &= x^2 + (a+b)x + ab \end{aligned}$$

### 4.10.2. Using packages

By default, Madoko supports most of the AMS mathematics commands, i.e. `amsmath`, `amsfonts`, and `amssymb`. However, sometimes you need specific packages. You can import more packages using the `Package` metadata key. For example, we can include the `amscd` package,

Package: `amscd`

to draw commutative diagrams:

```
~ Equation { #eq-commuting; caption:"A commuting diagram" }
\begin{CD}
S^{\mathcal{W}}_{\Lambda} \otimes T @>j>> T \\
@VVV @VV{P}V \\
(S \otimes T)/I @= (Z \otimes T)/J
\end{CD}
~
```

---


$$\begin{array}{ccc} S^{\mathcal{W}_{\Lambda}} \otimes T & \xrightarrow{j} & T \\ \downarrow & & \downarrow P \\ (S \otimes T)/I & \equiv & (Z \otimes T)/J \end{array} \quad (4)$$

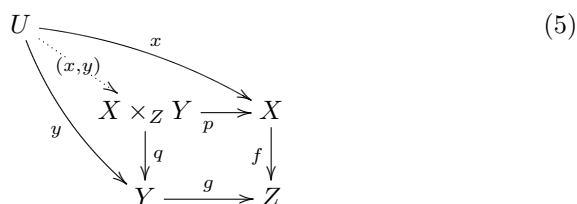
As a hint to understand the previous example, note that  $\textcircled{VVV}$  denotes a downward arrow,  $\textcircled{>j>>}$  a right pointing arrow with a label  $j$  on top, and  $\textcircled{=}$  a long equality.

Here is an example of the more modern [xypic](#) package which we included in this document with the `curve` option:

Package: `[curve]xypic`

With this package, we can draw more complex diagrams.

```
~ Equation { #eq-cat; caption:"Category theory" }
\xymatrix @-0.5em{
U \ar@/_/[ddr]_y \ar@/^/[drr]^x \\
\ar@{.>}[dr]|-{\langle x,y \rangle} \\
& X \times_Z Y \ar[d]_q \ar[r]_p \\
& X \ar[d]_f \\
& Y \ar[r]_g & Z }
~
```



As a hint to understand the above code, note that  $\text{\ar@/_/[ddr]}_y$  denotes an arrow, left curved ( $\textcircled{/_/}$ ), going down, down, right ( $\textcircled{[ddr]}$ ), with a label  $y$  underneath. See the package [user guide](#) for more examples.

**Madoko.net:** You can by default include any package included in [TexLive](#) (which are many). If you need to include a local package (or document class, bibliography etc) you can either use the toolbox menu or just drag&drop it into the editor.

### 4.10.3. Math commands

For math-heavy documents, it is convenient to define LaTeX commands for common operations. Such command definitions can be directly understood by LaTeX but need to be handled specially if the math is rendered dynamically in a HTML page. Madoko defines the custom block `MathDefs` to support mathematics definitions transparently across modes:

```

~ MathDefs
\newcommand{\infer}[3]{#1 \vdash #2\, : #3}
~
We infer  $\$ \infer{\Gamma}{e}{\tau} \$$  for such expression  $e$ .

```

---

We infer  $\Gamma \vdash e : \tau$  for such expression  $e$ .

Often it is convenient to put all such definitions in a separate `.tex` file and include it in the document as:

```

~ MathDefs
[INCLUDE="mathdefs.tex"]
~

```

#### 4.10.4. Mathematics in HTML

To typeset mathematics in HTML, Madoko can either typeset the math statically or dynamically:

- **static:** In the default static mode, Madoko uses LaTeX to generate static SVG (or PNG) images for each formula. This is generally preferred because pages load fast, and it requires no javascript. The drawback is that you need to have LaTeX installed on your system (or use [Madoko.net](https://madoko.net)). See the [metadata section](#) for more information on fine-tuning static mathematics.
- **dynamic:** Madoko uses the [MathJax](#) JavaScript library to render the math on the client-side. This means you do not need LaTeX but it is generally quite slow for math-heavy documents and not all of the LaTeX commands and packages are available. This mode can be enabled in the metadata as:

```
Math Mode: mathjax
```

The [metadata section](#) contains more information on customizing MathJax.

You can view and experiment with the different rendering modes in the online [mathematics rendering demo](#).

#### 4.10.5. Advanced: Generic LaTeX snippets

Besides just mathematics, you can include arbitrary LaTeX snippets using the custom `Snippet` block. This makes it possible to use powerful packages like [TikZ](#) and [PSTricks](#).

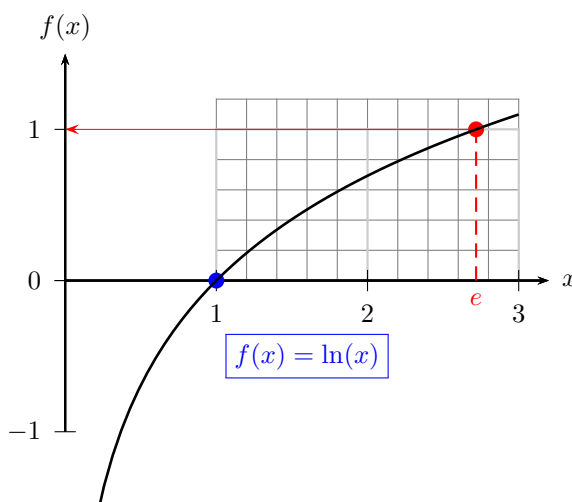
For the following example, we use PSTricks:

Package: pstricks

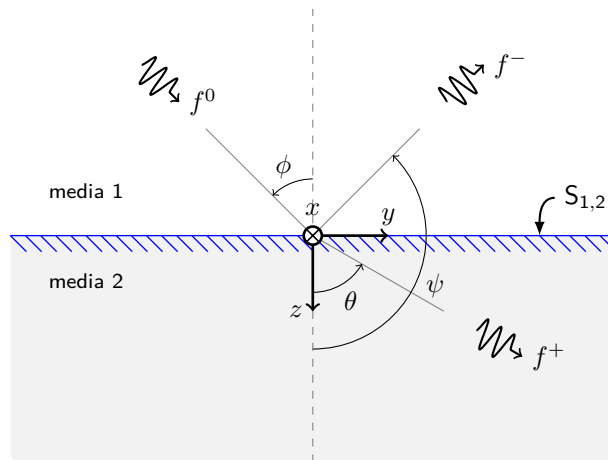
Package: pst-plot

to typeset a the graph of the natural logarithm<sup>4</sup>:

```
~ Center {padding:1ex}
~~ Snippet
\psset{unit=2cm}
\begin{pspicture*}(-0.5,-1.5)(4,2)
  \psgrid[subgriddiv=5,gridcolor=black!20%
    ,gridlabels=0pt](1,0)(3,1.2)
  \psaxes{->}(0,0)(0,-1)(3.2,1.5)
  \uput[0](3.2,0){$x$}\uput[90](0,1.5){$f(x)$}
  \pscircle*[linecolor=red](! Euler 1){3pt}
  \psline[linecolor=red,linestyle=dashed]%
    (! Euler 0)(! Euler 1)
  \psline[linecolor=red,linewidth=0.2pt,arrowscale=2]%
    {->}(! Euler 1)(0, 1)
  \uput[-90](! Euler 0){\color{red}$e$}
  \pscircle*[linecolor=blue](1,0){3pt}
  \psplot[linewidth=1pt]{0.2}{3}{ x ln }
  \rput(1.6,-0.5){\color{blue}\fbox{$f(x)=\ln(x)$}}
\end{pspicture*}
~~
~
```



<sup>4</sup>Example comes from the [PSTricks example gallery](#).



**Figure 3.** [TikZ example](#) by Edgar Fuentes of reflection and refraction of electromagnetic waves.

The next two figures, Figure 3 and Figure 4 use the modern [TikZ/Pgf](#) package which we included as:

```
Package      : tikz
Tex Header   : \usetikzlibrary{decorations.pathreplacing%
                  ,decorations.pathmorphing}
```

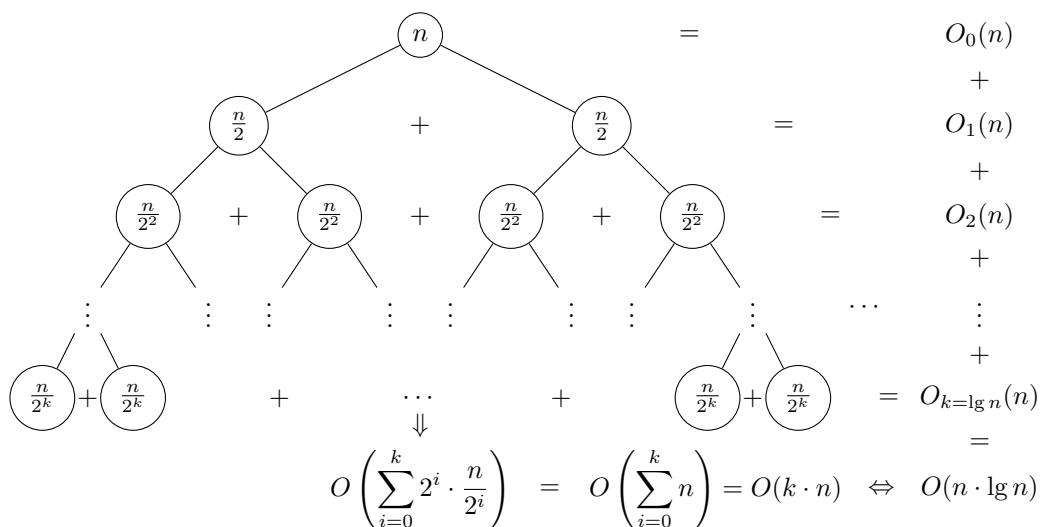
Both examples come from the [TikZ example gallery](#).

The `pgfplots` library is an extension of TikZ/Pgf to display plots:

```
Package: pgfplots
```

Drawing plots can be done either using math formulas or by giving direct data points:

```
Using the `pgfplots` package we can draw nice plots:
~ Snippet
\begin{tikzpicture}
\begin{axis}[
  height=8cm,
  width=8cm,
  grid=major,
]
% math plot
\addplot {-x^5 - 242};
```



**Figure 4.** [TikZ example](#) by Manuel Kirsch of a merge sort recursion tree.

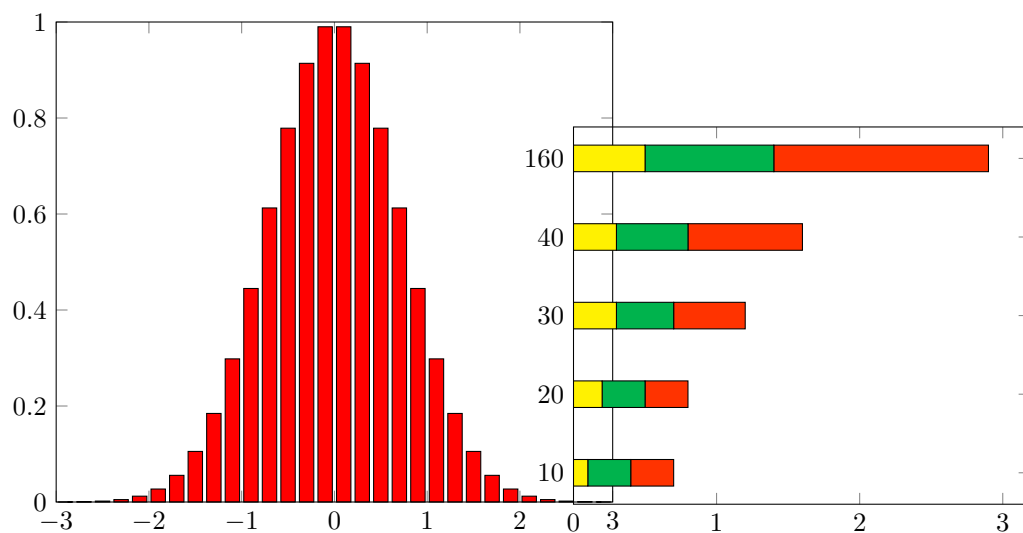
```

\addlegentry{model}
% data plot
\addplot coordinates {
(-4.77778,2027.60977)
(-3.55556,347.84069)
(-2.33333,22.58953)
(-1.11111,-493.50066)
(0.11111,46.66082)
(1.33333,-205.56286)
(2.55556,-341.40638)
(3.77778,-1169.24780)
(5.00000,-3269.56775)
};
\addlegentry{estimate}
\end{axis}
\end{tikzpicture}
~

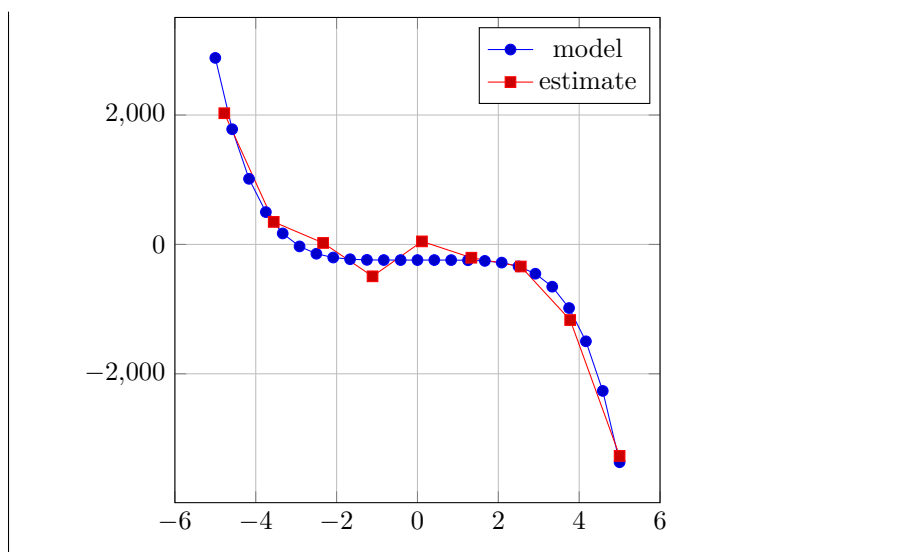
```

Using the `pgfplots` package we can draw nice plots:





**Figure 5.** Bar chart examples by [Matt B.](#) and [Jake](#) using `pgfplots`.



The final examples in Figure 5 also use the `pgfplotstable` library:

Package: `pgfplotstable`

to display bar charts.

### 4.10.6. Advanced: Efficiency of math rendering

Madoko is quite optimized to render mathematics efficiently, especially for math heavy documents which can easily contain thousands of small mathematics fragments. Madoko carefully processes only those fragments that need recompilation. If needed you can give the `-r` flag to rebuild everything from scratch (or say `Rebuild: True` in the metadata).

Usually, Madoko uses `xelatex` to generate a dvi (`.dvi`) file, and then uses `dvisvgm` to extract a scalable vector graphics (SVG) image, automatically taking care of proper baseline alignment. There are some cases where `dvisvgm` fails to extract a proper image – mostly for snippets that combine postscript and TikZ output drivers in `LATEX`; For those cases it can work to render them through `pdflatex` using the following metadata key (which was used for this document):

**Math Latex Full:** `pdflatex`

Sometimes snippets contain fragments that take a long time to process, in particular for certain TikZ programs. Madoko therefore has two independent modes, `plain` and `full`. By default Madoko uses for all regular mathematics the `plain` mode, and for `snippet`'s it uses `full` mode. This makes the preview more responsive when editing plain mathematics.

Certain snippets cannot work with plain dvi at all and need to be extracted as PNG files from a PDF file. In such cases, Madoko will use [ImageMagick's Convert](#) program to extract images. However, this is not done by default since this method is both much slower than extracting from a dvi file, and PNG images.

### 4.10.7. Advanced: Preformatted math

LaTeX math mode is great for regular mathematics but not so good if one tries to preserve whitespace or uses longer identifiers. This is actually quite common for computer science documents where mathematics is mixed with program code. Madoko supports a `MathPre` custom block that makes *preformatted* math much easier to typeset. In particular:

- Whitespace is preserved and spaces are replaced with a medium space (`\mathspace`) command (except when the spaces directly follow a LaTeX command). Indentation is done through a `\mathindent` command, while line breaks are automatic with a `\mathbr` command.
- A name (consisting of letters and digits) is typeset in a `\mathid` command so it will look like *function* instead of *function* (note the spacing between the *f* and *u* for example).
- If a name ends with digits, they are typeset as subscripts, where `x1` becomes  $x_1$ .
- A name starting with an `@` character is typeset using a `\mathkw` command, where `@return` becomes `return`.

- Any text that is an argument of a `\begin`, `\end`, `\text $xx$`  or `\math $xx$`  command, where  $xx$  is one of `tt`, `sf`, `sl`, `rm`, `it`, `kw`, `id`, `bb`, or `bf`, is kept unchanged. Also any name starting with a `#` character is kept unchanged.
- Ampersands can be used to align text.

Using this convention, we can easily typeset program code using nice symbols.

```
~ MathPre
@function sqr_\pi( num :int ) \{
  @return (num {\times} num {\times} \pi)
\}
~
```

---

```
function  $sqr_\pi$ ( num : int ) {
  return (num  $\times$  num  $\times \pi$ )
}
```

Here is an example of aligned text which also demonstrates the use of `replace` (see Section 5.6):

```
~ MathPre { replace: "/->/\rightarrow/g" }
random &: () -> ndet double;
print  &: string -> io ();
error  &: \forall\langle\alpha\rangle string -> exn \alpha;
~
```

---

```
random: ()  $\rightarrow$  ndet double;
print  : string  $\rightarrow$  io ();
error  :  $\forall\langle\alpha\rangle$  string  $\rightarrow$  exn  $\alpha$ ;
```

Of course, you can often achieve a similar effect by using good replacers with `.pretty` code directly (Section 4.8.3):

```
``` koka { .pretty; \
  replace: "//->/\(->|\&rarr;\)//</\(<|\&lang;\)//>/\(>|\&rang;\)//g"; \
```

```

        replace:"//\bforall\b\/\(\forall||&\forall;\)\)//\balpha\b\/\(\alpha|&\alpha;\)\)/"
random  : () -> ndet double;
print   : string -> io ();
error   : forall<alpha> string -> exn alpha;
...

```

---

```

random : () → ndet double;
print   : string → io ();
error   :  $\forall(\alpha)$  string → exn  $\alpha$ ;

```

#### 4.10.8. Setting all code to preformatted math

If you want to typeset all math using preformatted math, you can actually ‘take over’ the standard math blocks in Madoko and set the input of those to `mathpre`. For example:

```

.math-inline,.math-display {
  input: mathpre;
}

```

These set both display- and inline-math to the `mathpre` input mode.

### 4.11. Table of contents

Generating a table of contents is easy, just include the special element `[TOC]` anywhere in your document and it will expand to a table of contents. You can use the metadata value `Toc Depth` to set how many levels deep the table of contents goes (by default 3).

You can also create a list of figures or tables by using `[TOC=figures]` or `[TOC=tables]` respectively. This will list all the `~Figure` or `~TableFigure` block elements.

```

[TOC=figures]

```

---

# Figures

1. A Monarch butterfly.	22
2. Examples of sub figures.	24
(a) A butterfly . . . . .	24
(b) A matrix . . . . .	24
(c) A textual subfigure . . . . .	24
3. TikZ example by Edgar Fuentes of reflection and refraction of electromagnetic waves.	47
4. TikZ example by Manuel Kirsch of a merge sort recursion tree.	48
5. Bar chart examples by Matt B. and Jake using pgfplots.	49
6. The basic CSS colors	80
7. A formula	??

A [TOC] element comes with a heading by default, namely *Contents*, *Figures*, and *Tables*. The names can be customized using the metadata entities (§ 5.2) `name-contents`, `name-figures`, and `name-tables`.

## 4.11.1. Advanced: Custom tables of contents

Custom tables of contents can be generated using the `toc` attribute. The `toc-depth` attribute specifies the depth of the element (by default 1), while the `toc-line` specifies the contents of the line displayed in the table of contents. For example, using [metadata rules](#) we can generate a table of contents for equations:

```
equation {
  toc: equations;
  toc-line: "&label;\enspace&caption;";
}
```

Here we assume that the user adds a `caption` attribute when denoting equations (which will get expanded inside the `toc-line`). We can then render the table of equations anywhere in the document as:

```
## Equations { -; toc:clear; }
[TOC=equations]
✓
```

---

## Equations

(1) Euler's formula	21
(2) The Gaussian equation	41
(3) Alignment example	42
(4) A commuting diagram	43
(5) Category theory	44
(6) A chemical equation	100

## 4.12. Bibliography and Citations

One of Madoko's main design goals is to enable the creation of high-quality scholarly articles. As such, Madoko integrates closely with [BibTeX](#) and [Citation Style Language](#) (CSL) styles to generate bibliographies and references in a document.

You can simply use any existing BibTeX bibliography file (`.bib`) to describe all your references. You can specify which bibliography files are to be used using Bib (or Bibliography) [metadata](#) entries:

```
Bib: ../mybib1
Bib: mybib2
```

[Madoko.net](#): You can simply drag&drop any local `.bib` file into the editor to include it in the document.

A bibliography file consists of all your references in the BibTeX format, for example:

```
@BOOK{ Knuth:TeX,
  author   = "Knuth, Donald E.",
  title    = "The {\TeX} book",
  publisher= "Addison-Wesley",
  year     = 1984
}
```

You can view the bibliography file for this document [here](#). There is also a nice [table at Wikipedia](#) that shows all possible document types and field types.

Entries in the bibliography files can now be referenced using semi-colon sep-

arated references, for example

Read about LaTeX and TeX [[@Knuth:TeX](#); [@Lamport:LaTeX](#)].

---

Read about LaTeX and TeX [[4](#), [5](#)].

Note that unlike LaTeX there is no need to explicitly insert an unbreakable space between the text and the citation, Madoko automatically takes care of this (as described in Section 4.2). If necessary, you can also include extra text for each entry:

Read more [[The book @Knuth:TeX](#); \ [@Lamport:LaTeX \(chapter 4\)](#)].

---

Read more [[The book 4](#), [5 \(chapter 4\)](#)].

When Madoko finds such references, it writes them to a `.bib.aux` file (together with the needed bibliography files) and processes them to generate a bibliography. The generated bibliography entries are included in your document using the special `[BIB]` element.

[BIB]

---

## References

- [1] J. Fagerberg, D.C. Mowery, and R.R. Nelson, editors. *Oxford Handbook of Innovation*. Volume 1. Oxford University Press, Oxford. 2004.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley, Reading, Massachusetts. 1993.
- [3] O. Grandstrand. “Innovation and Intellectual Property Rights,” in Fagerberg et al. [1], volume 1, chapter 10. 2004.
- [4] Donald E. Knuth. *The T<sub>E</sub>X Book*. Addison-Wesley. 1984.
- [5] Leslie Lamport. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System (2nd Edition)*. Addison-Wesley. 1994. See also [4].

Note that a [BIB] element comes with a standard heading with the name *References*. You can customize this by setting the `name=references` entity.

### 4.12.1. Bibliography styles

The style of the bibliography entries is determined by a bibliography style. Madoko can use both standard [BibTeX styles](#) (`.bst`) and standard [Citation Language Style](#) styles (`.cs1`). An advantage of using CSL styles is that there are thousands available [online](#) and that they are processed internally in Madoko without needing to run BIB<sub>T</sub>E<sub>X</sub>.

CSL styles are included with the `Csl Style` key:

`Csl Style: Taylor-and-Francis-Chicago-Author-Date`

When using Madoko on the command line, you need to put the `taylor-and-francis-chicago-author-d` file in the document directory so it can be found by Madoko. There are three styles that come standard with Madoko. These styles handle cross references nicely and also handle DOI and URL fields well. The standard styles are:

- *madoko-numeric*: The default style which is a numeric style that closely follows the *plainnat* style if BIB<sub>T</sub>E<sub>X</sub>
- *madoko-numeric-short*: Similar to the previous style but follows *abbrvnat* with initialized authors and abbreviated terms.
- *madoko-natural*: Similar to the numeric style but for author-date style citations.



**Madoko.net:** If the CSL style is given without extension or directory, the style is looked up automatically from the CSL style [repository](#) and downloaded from there. You can also drag&drop a `.cs1` file in the editor pane to include it in the document.

When a CSL style is read, the citation style is automatically adjusted to reflect the CSL style, e.g. natural citations for author-date styles, or numeric citations for a numeric style.

Even with a CSL style, Madoko still expects the bibliography as a BibTeX `.bib` file – this is mostly because `.bib` files are so convenient to write and maintain and Madoko fully supports modern extensions like the `eprinttype` or `eprint` fields. However, if needed, one can also supply a `.json` file with CSL bibliography entries instead of a `.bib` file (by default, Madoko will actually generate such CSL bibliography from a `.bib` file in the output directory).

Instead of a CSL style, Madoko can also use any standard [BibTeX style](#):

**Bib Style:** `plainnat`

Madoko uses an internal LaTeX parser to format the bibliography entries in Markdown. It can handle things like special characters and accents quite well and recognizes many formatting commands. Even though it is sufficient for bibliography entries in general, the Madoko LaTeX parser may not work for more fancy LaTeX commands in bibliography entries. However, we strive to make it work for any bibliography style and entries, so please [file a bug report](#) if you encounter situations where it does not work correctly.

BibTeX bibliography styles tested with Madoko include the following author-year styles: `apa`, `apalike`, `plainnat`, `abbrvnat`, `unsrtnat`, `newapa`, `chicago`, `named`, `agsm`, `dcu`, `kluwer`, `astron`, `bbs`, `cbe`, `humannat`, `humanbio`, `jtb`, `apsrev4-1`, `aipauth4-1` and others. Also, the following numeric styles have been tested: `eptcs`, `abbrv`, `plain`, `ieeetr`, `acm`, `unsrt`, `alpha`, `siam`, `apsrmp4-1`, `aipnum4-1` and others. Since tools like BibTeX and Madoko make numbering and linking automatic, it is generally preferred for modern documents to use a numeric citations style.

### 4.12.2. Citation styles

The citation style determines how citations are shown in the document. It defaults to a *numeric* style, or the style set by a CSL style, but can be set explicitly using the **Cite Style** metadata key:

**Cite Style:** `natural`

Valid citation styles are *natural*, *sqnatural*, *textual*, *super*, and *numeric* (default). The natural and textual style use author-year style citations, while the super and numeric styles use numbers.

<i>natural</i>	(Lamport, 1994; Knuth, 1984)	(default for author-year citations)
<i>sqnatural</i>	[Lamport, 1994; Knuth, 1984]	
<i>textual</i>	Lamport (1994); Knuth (1984)	
<i>numeric</i>	[4, 5]	(default for numeric citations)
<i>super</i>	<sup>4,5</sup>	

Note that numeric citations are sorted (and compressed) by default. Also full author-year style citations only work with CSL styles, and with BibTeX styles that support this, i.e. generally any style that works with the `natbib` package like `plainnat`. With author-year citations we can use modifiers to change how the citation is shown. For example, assuming a *natural* style:

<code>[@Goo93]</code>	(Goossens et al., 1993)	Natural
<code>[+@Goo93]</code>	(Goossens, Mittelbach, and Samarin, 1993)	Long – all authors
<code>[-@Goo93]</code>	(1993)	Short – just year

Moreover, if you leave out the brackets, you force a *textual* style:

<code>@Goo93</code>	Goossens et al. (1993)	Textual style
<code>+@Goo93</code>	Goossens, Mittelbach, and Samarin (1993)	Long – all authors
<code>-@Goo93</code>	1993	Short – just year
<code>!@Goo93</code>	Goossens et al.	Just authors

In a numeric style, most of these attributes have no effect and all the bracketed cases are displayed as “[2]”. For the four cases in textual style, we get “Goossens et al. [2]”, “Goossens, Mittelbach, and Samarin [2]”, “2”, and “Goossens et al.”.

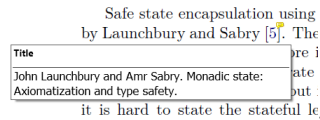
See Appendix A.3 on how to customize citations styles further, and Appendix A.4 on how to write your bibliography entries by hand without using a `[BIB]` element.

### 4.12.3. Bibliography tooltips and searches

By default, the HTML backend shows a tooltip when hovering over a citation (try it [2]). This functionality is by default disabled in the LaTeX backend. However, you can enable tooltips in PDF too by setting the `.tex-tooltip` class through a metadata rule:

`~a: .tex-tooltip`

This will display a small yellow text balloon in the PDF file that shows a tooltip when hovering above it:



Moreover, Madoko adds to each bibliography entry a magnifying glass icon (🔍) that links to a web search for that reference. You can customize the search engine that is used by setting the `Bib Search Url` metadata key:

`Bib Search Url: www.google.com`

If you set the `Bib Search Url` to `false` (or empty), Madoko will disable search icons. Again, this functionality is disabled by default in the LaTeX backend unless you set the `Bib Search Url` key explicitly

## 4.13. Custom blocks

Madoko custom blocks are similar to the `div` element in HTML and allow the use of custom block elements that can be styled and processed in a particular way. A custom block starts on new line starting with one or more tildes (`~`) optionally followed by the block name and attributes. It ends at *the first line containing the same number of tildes* that started this block.

```
~ Note
Here is a note.
~
~~ { font-style: italic }
And some italic text in an unnamed block.
~~
```

---

**Note.** Here is a note.  
*And some italic text in an unnamed block.*

Note that blocks with the same number of tildes do not nest, e.g. the following is wrong:

```
~ Note
~ Equation
e = mc^2
~
~
```

since the **Note** will end at the first lonely tilde, not the second one. As an aside, [git flavored markdown](#) uses three or more tildes for fenced code blocks. Since Madoko uses tildes for custom code blocks this cannot be used and Madoko only supports the more popular back-ticks (`````) for fenced code blocks.

Custom blocks work especially well with metadata rules (see [Section 5.4](#)) where we can define attributes that get applied to every occurrence of a custom block. For example, we could define the metadata rule:

```
slanted {
  font-style: oblique;
}
```

and then every occurrence of a **Slanted** custom block would be typeset in a slanted font.

```
~ Slanted
Here is my slanted custom block
~
```

---

*Here is my slanted custom block*

The “number-of-tildes” rule to delimit custom blocks is convenient and works fine when nesting a small number of custom blocks, but for long blocks or deep nesting, this can easily lead to confusion. To alleviate this, a custom block can also start with one or more tildes, followed by **Begin** *blockname*. Such block continues until a corresponding number of tildes is found followed by **End** *blockname*. It is recommended to use this form of blocks for example in [metadata rules](#)

```
~ Begin Slanted

~ Begin Note
Here is a slanted note.

~ End Note
~ End Slanted
```

---

**Note.** *Here is a slanted note.*

#### 4.13.1. Predefined custom blocks

Madoko defines quite a few common custom blocks. Their exact definitions can be found in [Appendix A.10](#).

- **Figure:** This is used to define figures (see [Section 4.3](#)). Recognizes the following attributes:
  - **caption:** *caption*: specify the caption of a figure.
  - **.wide:** if the class **wide** is set, the figure will span the width of a page in a two-column format (used in LaTeX).
  - **page-align:** (*top|bottom|topbottom|page|here|forcehere|inplace*): specify the alignment of the figure in a page (used for LaTeX output). **forcehere** corresponds to the **h!** specifier in LaTeX, while **inplace** uses **H**. If you need even more customization, you can set the specifier in LaTeX explicitly using the **tex-float-placement** attribute.
- **Equation:** Specify mathematical equations. See [Section 4.10](#) for its usage.

- **Snippet**: Generic LaTeX snippets, see Section 4.10.5.
- **Bibitem, Bibliography**: Used for writing bibliography entries by hand. See Section 4.12 for more information.
- **Note, Remark, Proof**: Used for notes, remarks, and proofs.
- **Abstract**: Defines the abstract of an article.
- **Framed**: A block with a solid border. Use the `tight` attribute to suppress a paragraph block around its content.
- **Center**: A block that centers its contents horizontally on the page.
- **Columns, Column**: Put `Column` blocks inside a `Columns` block, to typeset block elements next to each other on a page. Use the `width` attribute to control the width of each column.
- **TexRaw**: A block for raw TeX content that is passed directly to LaTeX.
- **Section**: Starts an explicit section. In the HTML5 backend expands to a `<section>` element. This is mainly used inside presentations using `reveal.js`.
- **Article, Aside, Nav**: Expand to the corresponding HTML5 elements.
- **HtmlRaw**: A block with raw HTML content that is pasted directly into the HTML output. This is normally not needed since you can just start including HTML elements directly as part of markdown input.
- **TexOnly**: A markdown block that is only processed for LaTeX output (and not shown in HTML output).
- **HtmlOnly**: A markdown block that is only processed for Html output (and not shown in LaTeX output).
- **MathPre**: A block with preformatted math (Section 4.10.7).
- **MathDefs**: A block with LaTeX math definitions (Section 4.10.3).
- **Theorem, Lemma, Proposition, Corollary, Example, Definition**: Each of these blocks is individually numbered and starts with the block name in bold. All of these can take a `caption` attribute. For example:

```
~ Lemma      {#LeftCosetsDisjoint; caption: "Left co-sets are disjoint"; }
Let  $H$  be a subgroup of a group  $G$ , and let  $x$  and
 $y$  be elements of  $G$ . Suppose that  $xH \cap yH$  is
non-empty. Then  $xH = yH$ .
~
```

```

~ Proof { caption: "Of Lemma [#leftcosetsdisjoint]" }
Let  $z$  be some element of  $xH \cap yH$ . Then  $z = xa$ 
for some  $a \in H$ , and  $z = yb$  for some  $b \in H$ .
If  $h$  is any element of  $H$  then  $ah \in H$  and
 $a^{-1}h \in H$ , since  $H$  is a subgroup of  $G$ .
But  $zh = x(ah)$  and  $xh = z(a^{-1}h)$  for all
 $h \in H$ .
Therefore  $zH \subset xH$  and  $xH \subset zH$ , and thus
 $xH = zH$ . Similarly  $yH = zH$ , and thus  $xH = yH$ ,
as required. &qed;
~

```

---

**Lemma 1.** (*Left co-sets are disjoint*)

Let  $H$  be a subgroup of a group  $G$ , and let  $x$  and  $y$  be elements of  $G$ . Suppose that  $xH \cap yH$  is non-empty. Then  $xH = yH$ .

**Proof.** (*Of Lemma 1*) Let  $z$  be some element of  $xH \cap yH$ . Then  $z = xa$  for some  $a \in H$ , and  $z = yb$  for some  $b \in H$ . If  $h$  is any element of  $H$  then  $ah \in H$  and  $a^{-1}h \in H$ , since  $H$  is a subgroup of  $G$ . But  $zh = x(ah)$  and  $xh = z(a^{-1}h)$  for all  $h \in H$ . Therefore  $zH \subset xH$  and  $xH \subset zH$ , and thus  $xH = zH$ . Similarly  $yH = zH$ , and thus  $xH = yH$ , as required.  $\square$

Of course, each of the predefined blocks can be customized further using attributes and rules. For example, by including the following metadata rule, we can typeset proofs with *Proof* in an italic style instead of bold:

```

Proof {
before: "[_Proof_. ]{.proof-before}"
}

```

## 4.14. Special block elements

Madoko supports some special block elements that get expanded automatically.

- [TITLE]: Expands to a title element generated from the `metadata` keys *title*, *subtitle*, *title date* and author info (*author*, *affiliation*, and *email*). In the LaTeX backend this will invoke the `\maketitle` command.
- [TOC], [TOC=*name*]: Expands to a table of contents. See also Section 4.11.

- `[FOOTNOTES]`: Expands to the defined footnotes. If there are footnotes and this element is not present, the footnotes are automatically included at the end of the document. In the LaTeX backend, footnotes always appear at page footers.
- `[INCLUDE=file]`: Expands to the contents of *file*. Include elements are processed before any other processing happens and can for example be used to include meta data elements.

```
[INCLUDE="presentation"]
```

Include files are first searched relative to the current directory, then the directory of the input file, followed by the output directory, and finally the `styles` directory in the installation directory of Madoko. The default file extension is `.mdk`.

The `INCLUDE` element can also be used to include parts of a file, see Section 4.14.1 for more information.

- `[BIB]`: Includes the bibliography entry file (`.bbl`) generated by BibTeX. Internally expands to:

```
~ Begin TeX
[INCLUDE="myfile.bbl"]
~ End TeX
```

See also Section 4.12.

#### 4.14.1. Advanced: including file fragments

The full syntax of the `INCLUDE` element is:

```
[INCLUDE(=file)?(:range)?]
```

where the *range* is either a *fragment name* or line range. For example we can include part of file by using:

```
[INCLUDE=foo.hs:10-18]
```

or

```
[INCLUDE=foo.hs:20]
```

which would include lines 10 to 18, and all lines starting at line 20 from `foo.hs`. Of course remembering line ranges can be error prone, so we can also give names to fragments of files and use those to refer to them. For example, we may have code samples in `foo.hs` as:

```
-- BEGIN:Var
-- BEGIN:Syntax
data Exp = Number      Int
        | Add          Exp Exp
        | Subtract     Exp Exp
```

```

        | Multiply  Exp Exp
        | Divide    Exp Exp
-- END:Syntax
        | Variable  String;
-- END:Var

--BEGIN:Eval
evaluate :: Exp -> Int
evaluate (Number i)      = i
evaluate (Add a b)        = evaluate a + evaluate b
evaluate (Subtract a b)   = evaluate a - evaluate b
evaluate (Multiply a b)   = evaluate a * evaluate b
evaluate (Divide a b)      = evaluate a `div` evaluate b
--END:Eval

```

Where the `BEGIN name` and `END name` delimit file fragments. Note that the fragments can be nested and/or overlapping. If a fragment is not ended, it will run until the end of the file. When fragments are included, and fragment begin/end lines are replaced by empty lines. We can now include particular fragments as:

```

[INCLUDE=foo.hs:Var]
[INCLUDE=foo.hs:Eval]

```

for example. Sometimes, it can be a hassle to always mention the full file name. In that case, we can first do a few empty includes of all the needed files (so Madoko knows the fragment names), and then just use the fragment names directly, e.g. first an empty include at the start of the document by using line number 0:

```

[INCLUDE=foo.hs:0]

```

and later in the document, we can refer to fragments in that file using:

```

[INCLUDE:Syntax]

```

The delimiters for fragments are always ‘*start* (BEGIN|END) *name*’ where *start* includes most of the usual comment characters, namely, `//`, `--`, `<!--`, `(*`, `#`, and `%`. If you need some other sequence, you can customize this using the `Fragment Start` and `Fragment End` metadata keys. For example, the default is set as:

```

Fragment Start: ^(?:\\\/|--|[#%]|<!--|\\(\\*) *BEGIN *: *(\\w+) *(?:-->|\\*\\))?$
Fragment End   : ^(?:\\\/|--|[#%]|<!--|\\(\\*) *END *\\
                (?:[:] *(\\w+) *)?(?:-->|\\*\\))?$

```



## Chapter 5

# Syntax: Metadata and Styling

### 5.1. Metadata

Similar to [multimarkdown](#), a document can begin with a special metadata section that contains meta information like the document title, the author, etc. Moreover, this section can contain attribute rules to globally apply attributes to certain elements, much like CSS rules.

Metadata must come immediately as the first thing in a document, and consists of keys followed by a colon and then the key value. A key value can span multiple lines by indenting, and you can leave blank lines between different keys.

```
Title      : An overview of Madoko
Author     : Daan Leijen
Affiliation : Microsoft Research
Email      : daan@microsoft.com
Logo       : False
Embed      : False
```

In general, metadata keys are not case-sensitive. The values `none`, `clear`, `false`, and the empty value are all equivalent for boolean or numeric keys.

#### 5.1.1. Special metadata keys

Any metadata key and value can be given (and referred to using [entity names](#)), but certain keys have special meaning to Madoko.

- **Title:** The title of the document. For example, in HTML output it determines the `<title>` element. It is also used by the special `[TITLE]` element (Section [4.14](#)).

- **Subtitle:** An optional subtitle.
- **Title Note:** A general note about the document. Used in the `[TITLE]` element where it is put below the title and authors.

`Title Note: Draft, &date; (version 1.0)`

- **Title Footer:** A general footer that is placed below the title and authors.
- **Author, Address, Affiliation, Email, Author Note:** the author name, address, affiliation, email, and general note. There can be multiple authors. The author info is also used by the `[TITLE]` element to generate a proper document title header. The authors are also written to the author `<meta>` tag. In the case of multiple authors, the address, affiliation, author note, and email always belong to the last defined author. If you define multiple **Address** or **Affiliation** entries for one author, they are placed above each other.
- **Title Running, Authors Running:** Used for certain LaTeX styles as the running title and authors displayed at the top of each page. By default the full title and list of authors are used.
- **Author Columns (=3):** The number of columns used for the authors. Only relevant if there are more than 3 authors used.
- **Toc depth (=3):** The maximum depth of headings that are included in the table of contents (Section 4.11).
- **Heading depth (=3):** The maximum depth of headings that are numbered (Section 5.5). Set it to zero to suppress numbering of headings completely.
- **Heading base (=2):** Usually, a top heading (`# heading`) maps to a `<h2>` or `\section` element in HTML and LaTeX respectively. By changing the **Heading Base** you can change this mapping. For example, by using:

`Heading Base: 1`

the top headings will map to `<h1>` or `\chapter` elements instead.

- **Section Depth (=0):** Maximal heading level where `<section>` elements are wrapped around the content. The default value ensures no section wrapping takes place.
- **Section Base (=1):** All headings at **Section Base** and lower are treated as equal for the purpose of wrapping `<section>` elements around the content. This is especially useful for presentations where we would like to start slides equally for headings at level 1 and 2 (see Appendix A.2).
- **Bib or Bibliography:** Specify a bibliography (`.bib`) file to be used by the BibTeX tool to generate a list of references (see Section 4.12).

`Bib: ../mybibliography.bib`

- **Bib Style** or **Biblio Style** (=plain): Specify a BibTeX style that is used to format the list of references. See Section 4.12.1 for more information.

**Bib Style:** plainnat

[Madoko.net](#): When leaving out the extension, Madoko assumes this is a standard bibliography style (available from CTAN). If an extension (**.bst**) is given, the file is loaded (or created) from the current directory.

- **Csl Style**: Specify a Citations Language Style (CSL) that is used to format references. This takes precedence over a **Bib Style** since CSL styles are generally more up-to-date and often have better localization.

**Csl Style:** ieee

[Madoko.net](#): When leaving out the extension and directory, Madoko assumes this is a standard bibliography style and will automatically download it from the [CSL style repository](#).

Also, CSL styles will be handled completely client-side without the need for a server roundtrip.

- **Locale** (=english): Specify the document locale, either by the language name (**german**), or as language identifier (like **en-US** or **fr-CA**). Currently not used by Madoko but it is used when generating bibliographies using a CSL style for that locale.
- **Cite Style** (=numeric): Specify the citation style used for citations, *natural*, *\_textual*, *super*, or *numeric*. See Section 4.12.2 for more information.
- **Cite All** (=false): Set to **True** to cite everything in the included bibliography files.
- **BibTex** (=bibtex): The command to run the BibTeX tool. Set it to **false** to suppress running BibTeX (for example, when writing bibliography entries by hand (see Section A.4))
- **Pdf Latex** (=xelatex): The command used to generate a PDF when the **--pdf** command line flag is present.
- **Latex** (=latex): The command used to generate a DVI when necessary (e.g. when generating plain math with PNG images)
- **Fragment Start**, **Fragment End**: Used to specify file fragment delimiters, see Section 4.14.1.
- **Logo** (=True): Set to **False** to suppress the “Document generated by Madoko.net” message at the end of the document.
- **Prelude** (=prelude): Specify the standard prelude style that is always included.

- **Refer:** Make explicit that a certain file or image is referenced by this document.

[Madoko.net](#): This is useful in the online editor if the inclusion of the file is not apparent to Madoko. This can happen for example in LaTeX class files that reference images, or image references inside HTML attributes.

- **Line No (=1):** Specify the starting line number used for errors and warnings. Usually, the generated HTML and LaTeX will contain these line numbers to track back errors to the original sources. Set **line-no** to **False** to suppress line number generation.
- **Pretty Align (=2):** The number of spaces needed to align code in **.pretty** mode.

The following keys are not used as this moment but have a standard meaning:

- **Copyright:** Copyright information. Written to the copyright **<meta>** tag.
- **License:** License for this document. Written to the license **<meta>** tag.
- **Keywords:** A list of document keywords. Written to the keywords **<meta>** tag.
- **Description:** A short description of the document. Written to the description **<meta>** tag.
- **Comment:** General comments.
- **Revision:** Revision of the document.
- **Phone:** Phone number of an author.

### 5.1.2. HTML keys

Some keys are only interpreted for HTML output:

- **Css:** Specify a CSS file that needs to be included in HTML output. There can be many CSS keys present.

```
Css: lib/main.css
Css: http://foo.com/bar.css
Css: http://fonts.googleapis.com/css?family=Open+Sans
```

Use **Css: clear** to clear the list of CSS files. This can be used for example to not include the default **madoko.css** style file.

[Madoko.net](#): When not giving an extension or full path, Madoko will try to load the css file from the Madoko server's **/style** directory. This is used for the standard **madoko.css** file.

- **Script:** Specify a Javascript file that needs to be included in the HTML output via a **<script>** tag. There can be many script keys present.

```
Script: lib/main.js
```

[Madoko.net](#): Usually, scripts are *not* loaded in the preview as this may lead to performance problems. You can use the `preview` class to signify that the script should be loaded in the preview too:

```
Script: reveal.js {.preview}
```

- **Embed (=512)**: The file size limit in kilobytes up to which local data is directly embedded into the HTML, this includes CSS style files, Javascript files, and images. Set it to 0 to never embed anything, and high enough to always embed.
- **HTML Meta**: Specify content included in a `<meta>` tag. There can be many meta tags specified.

```
HTML Meta: http-equiv="refresh" content="30"
```

Use the value `clear` to clear the list of meta tags. This can be used for example to exclude the default `viewport` meta tag that is emitted for proper viewing on mobile devices.

- **HTML Header**: The value is included literally in the `<head>` section of the HTML document.
- **HTML Footer**: The value is included literally at the end of the HTML document.
- **CSS Header**: The value is included literally in a `<style>` element in the `<head>` section of the HTML document.
- **JS Header**: Literal Javascript that is included in a `<script>` element in the `<head>` section of the HTML document.
- **JS Footer**: Literal Javascript that is included in a `<script>` element at end of the HTML document.
- **Math Mode: static|mathjax|dynamic (=static)**: The mathematics rendering mode is either `static` or `dynamic`. You can set the **Math Mode** to `mathjax` to set the mode to `dynamic` and `Mathjax` to `True`. In static mode, you can use the following keys to fine-tune the rendering:
  - **Math Dir: dir (=math)**: The directory relative to the output directory where Madoko stores the images for the math formulas. All math images can also be embedded in the web page if you set **Math Embed** high enough.
  - **Math Embed:size (=512)**: Limit in kilobytes at which images are embedded in the HTML page instead of separate `.svg` (or `.png`) images. Set it to 0 to never embed an image, and high enough to always embed.

- **Math Scale:** *scale*: Scaling in percentage used for all math. The default looks quite good for most fonts, but you may want to change it depending on the main font used. The default used is `&Math Scale Svg`; (105) for SVG math images, and `&Math Scale Png` (108) for PNG images.
- **Math Latex:** *cmd*: The command used to invoke LaTeX when rendering plain math. When extracting SVG it defaults to `&Pdf Latex`; and when extracting PNG to `&Latex`;
- **Math Latex Full:** *cmd*: The command used to invoke LaTeX when rendering math that requires a PDF; defaults to `&Pdf Latex`;
- **Math Render, Math Render Full:** *svg|png (=svg)*: Determines whether to generate scalable vector graphics (**svg**) for mathematics or hi-resolution bitmaps (**png**). The SVG images look much better in general and scale automatically to any size.

When generating a SVG the following options are available:

- \* **Math Scale Svg:** *scale (=105)*: Scaling in percentage used for math extract to a SVG images. The default looks quite good for most fonts, but you may want to change it depending on the main font used.
- \* **Math Svg Precision (=3)**: Number of decimal digits of precision in the image (as a fraction of a **pt**). Increasing this makes the paths more precise but can increase the final size of the generated HTML.
- \* **Math Svg Share Paths (=True)**: If true, Madoko will share all common graphics paths among different math formulas. This can compress the size of the final HTML significantly since every individual glyph inside formulas will only be defined once.
- \* **Math Svg Use Fonts (=False)**: When true, all the text in a math formula will be included as text with a proper font reference. When false, each font glyph is statically traced as graphics path; the latter is less efficient but portable across systems even if fonts are not available. At at this time (2015) most browsers usually do not have good direct font usage support and it usually comes out looking quite bad.
- \* **Dvisvg:** *cmd (=dvisvgm)*: Specify the **dvisvgm** program used to generate **svg**'s from a **.dvi** (or **.xdv**) file. Madoko requires at least version 1.14 of **dvisvgm** to work correctly; if Madoko finds a lower version it switches automatically the rendering mode to PNG. The latest version is available at the [dvisvgm website](#) where versions are available for Linux, MacOSX, and MikTeX. For Windows with TexLive 2015 you can extract an executable from the [w32tex.org](#) website. Since this is somewhat involved, you can also download a [pre-compiled binary \(win32,v1.15\)](#) from the Madoko website and put it in the `c:\texlive\2015\bin\win32`

folder.

- \* **Math Dpi:** *dpi* (=300): Does not apply directly to scalable vector graphics since these can render at any scale. However, this setting is still taken into account for rebuilding all mathematics. By changing the value a little you can force rebuilding all math.

When generating **png** images, the following options apply:

- \* **Math Dpi:** *dpi* (=300): The resolution at which the images are rendered. The default is a fairly high resolution. Decrease it to make the generated images smaller. Use the `-vv` flag to see the sizes of all the images generated for formulas.  
*Note:* you can also use this setting to easily rebuild all the mathematics in your document by changing the value little bit.
- \* **Math Scale Png:** *scale* (=108): Scaling in percentage used for math extract to a **.png** image. The default looks quite good for most fonts, but you may want to change it depending on the main font used.
- \* **Convert:** *cmd* (=convert): Specify [ImageMagick's convert](#) command that is executed for images that are generated from PDF output. By default, any *full* math uses **convert** to extract from PDF.
- \* **Dvipng:** *cmd* (=dvipng): The command used to extract **.png** images from the **.dvi** files generated by LaTeX. By default, any *plain* math is processed using **latex** and uses **dvipng** to extract images.
- \* **Dvips:**, **Ps2pdf:** *cmd*: These commands are used for *full* math if the key **Math Latex Full** is set to plain **latex** in which case Madoko needs to extract a PDF from the DVI file.

The following keys are used when math mode is dynamic:

- **MathJax** (=false): Set this key to the path of your MathJaX script. Set this key to **True** to include the standard secure script to the latest MathJaX installation on the [CDN network](#).

**MathJax:** **True**

- **MathJax Ext:** requires that the **MathJax** key is set. Adds a MathJaX extension to the loaded scripts. See Section [A.5](#) for more information.

### 5.1.3. LaTeX keys

For LaTeX output, the following keys are relevant:

- **Document Class** or **Doc Class** (=book): Specify the LaTeX document class. Can be prefixed with its options using square brackets:

Document Class: `[9pt]article`

[Madoko.net](#): When leaving out the extension, Madoko assumes this is a standard LaTeX document class (available from CTAN). If an extension is given, the file is loaded (or created) from the current directory.

- **Package**: Specify a LaTeX package that is included via `\usepackage`. The package name can be prefixed with its options using square brackets.

Package: `fancyhdr`

Package: `[colorlinks=true]hyperref`

If the package name ends with `.tex` and has no options, it will be included using the `\input` command instead. You can clear the package list using **Package: clear**. This can be used to not include the default `madoko2.sty` package.

Sometimes, a package does not work when typesetting mathematics in plain LaTeX. This can happen for example if the rest of the document needs a package that only loads in some other latex engine, like XeLaTeX. In that case, you can exclude the package when typesetting basic math by using the star option:

Package\*: `pgfplots`

[Madoko.net](#): When leaving out the extension, Madoko assumes this is a standard LaTeX package (available from CTAN). If an extension is given, the file is loaded (or created) from the current directory.

- **Tex Header**: The value is included as is before the `\begin{document}` command of the LaTeX output.
- **Tex Header\***: The value is included as is before the `\begin{document}` command of the LaTeX output but excluded when rendering basic mathematics.
- **Tex Doc Header**: The value is included as is right after the `\begin{document}` command of the LaTeX output.
- **Tex Doc Header\***: The value is included as is right after the `\begin{document}` command of the LaTeX output but excluded when rendering basic mathematics.
- **Maketitle (=True)**: If `True`, the `\maketitle` command is used to typeset the title and authors instead of using the standard Madoko styling.
- **Bib Label:(False|True|Keep|Hide) (=False)**: Signifies how the bibliography items are labeled by L<sup>A</sup>T<sub>E</sub>X. By default the L<sup>A</sup>T<sub>E</sub>X label is suppressed. By setting **Bib Label** to `True`, the default L<sup>A</sup>T<sub>E</sub>X is used. `Keep` resets the bibliography label to the default L<sup>A</sup>T<sub>E</sub>X definition while `Hide` suppresses



the  $\text{\LaTeX}$  label but keeps the whitespace. Some styles and packages in  $\text{\LaTeX}$  require sometimes one these variants. For example, the standard L<sup>A</sup>T<sub>E</sub>X style in [Madoko.net](https://madoko.net) uses `Bib Label: True` to render the bibliography item labels in the right style required by L<sup>A</sup>T<sub>E</sub>X document class.

#### 5.1.4. Conditional metadata

Sometimes it is convenient to apply a certain metadata rule only for specific output formats or specific situations. For this you use a conditional block. For example, to only add a logo when generating HTML, you can say:

```
@if html {
  Logo: True
}
```

You can standard [CSS conditional rules](https://madoko.net). To make all subsections blue except when generating a PDF, you can say:

```
@if not tex {
  h2 { color: blue }
}
```

Any metadata value can be referenced in the conditional expressions. Some useful entities are defined by Madoko by default:

- `html`: True when generating HTML output.
- `tex`: True when generating LaTeX/PDF output.
- `preview`: True when generating HTML in the [Madoko.net](https://madoko.net) preview.

You can also create your own metadata values, for example:

```
draft: True
```

```
@if not draft {
  Todo { display: none }
}
```

## 5.2. Entities

Madoko uses entities extensively. If Madoko finds an entity name (`&name;`) it looks up the name in various places: if there is a block element with that *id*, the entity name is replaced by the *label* value of that block (see Section 4.2). If there is no block with that identity, Madoko looks if there is a metadata value with that name, and replaces the entity name with its value:

```
The title of this document is "&title;".\
And this section has label &sec-entity;.\
```

Standard entities are looked up last `&Delta;``&hArr;``&delta;`.

The title of this document is “Madoko Reference”.

And this section has label 5.2.

Standard entities are looked up last  $\Delta \Leftrightarrow \delta$ .

Entity names for labels and metadata values, can consist of letters, underscores, minus signs, and digits. In contrast to regular HTML entity names, Madoko compares the entity name for labels and metadata in a case-insensitive way.

Note that if label or metadata values contain themselves entities, these are expanded recursively (up to a certain limit) which can provide a powerful abstraction mechanism. For example, if we define a metadata value `Title2`:

`Title2: (&Title;,&Title;)`

then the entity `&Title2;` expands to “(Madoko Reference,Madoko Reference)”.

If entity names are used inside `attributes`, Madoko first looks for the name as one of the attribute key values. For example, you can use names like `&id;`, `&class;`, and `&label;`.

Some predefined metadata keys are quite useful as entities:

- `&date;`. The current (compilation) date in [ISO 8601](#) international format.
- `&time;`. The current (compilation) time in [ISO 8601](#) international format.
- `&year;``&month;``&day;`. The current year, month, and day.
- `&hours;``&minutes;``&seconds;`. The current hours, minutes, and seconds.
- `&madoko-version`. The version of the Madoko compiler.
- `&docname;`. The file name of the document without extension and directory.
- `&filename;`. The full file name of the document.
- `&eg;`. Expands to “e.g.” (“*exempli gratia*”, “for example”).
- `&ie;`. Expands to “i.e.” (“*id est*”, “that is” or “in other words”).
- `&etal;`. Expands to “et al.” (“*et alii*”, “and others”).
- `&logotex;`. Expands to “ $\text{T}_\text{E}\text{X}$ ”.
- `&logolatex;`. Expands to “ $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ ”.
- `&logolatexe;`. Expands to “ $\text{L}_\text{A}\text{T}_\text{E}\text{X}_2\epsilon$ ”.
- `&logobibtex;`. Expands to “ $\text{BIB}\text{T}_\text{E}\text{X}$ ”.

There are also a few special entity names:

- `&nl;`. Expands to the newline character.
- `&br;`. Expands to a forced line break (i.e. `<br>`).
- `&>null;`. Expands to an empty string.
- `&source;`. Inside attributes, expands to the literal content of the element.
- `&&`. Expands to the literal `&` character. This is necessary sometimes since entity expansion is done recursively without regard for other formatting attributes. For example, if you have a metadata value `Foo` that needs to expand to ``&amp;``, you would need to write it as:

```
Foo: `&&amp;`
```

or otherwise the `&amp;amp;` part would get expanded even within the back ticks. The reason why Madoko expands regardless of other formatting is to allow powerful abstraction where we can for example build up regular expressions from smaller parts.

For some examples of the usage of these elements, see Sections 5.6 and 5.5.2.

## 5.3. CSS Attributes and Styling

Writing Madoko documents is clearly about content: the main advantage of using a markdown format is that it allows the writer to concentrate on prose and content instead of formatting. However, in the final stages it is desirable to *style* a document to make it look good. This section describes some tricks and tips that can help doing this well.

### 5.3.1. CSS Attributes

An essential addition of Madoko is good support for attributes. The syntax is like CSS attributes and denoted between curly braces:

```
{ key: value; key: value }
```

Moreover, there are two convenient shorthands for names (`id:name`) and classes (`class:classnames`) using a `#` and `..`:

```
{ .class; #id; key: value }
```

For most block elements, like lists, paragraphs, block quotes, etc, you can write attributes on the line directly following the block. For list items, the attributes directly follow the item:

```
This is a paragraph in small-caps.
{ font-variant: small-caps}
```

```
* {color: navy} This is a 'mylist'
* in italics
{ .mylist; font-style: italic }
```

---

```
THIS IS A PARAGRAPH IN SMALL-CAPS.
```

- *This is a 'mylist'*
- *in italics*


For fenced code blocks, custom blocks, and headers, the attributes are specified on the first line.

For inline elements, you can follow links, images, code, and bracketed text (between [ and ]) with attributes. Bracketed text with attributes become a `<span>` in the HTML backend. It is of course recommended to put attributes when possible in link or image definitions themselves:

```
Here is an ![butterfly] image.
With [bold]{font-weight:bold} text, and even the
T[E]{vertical-align: -0.5ex; margin-left: -0.25ex; \
    margin-right: -0.25ex}X
logo.

[butterfly]: images/butterfly-200.png { width: 100px; vertical-align: top }
```

---

Here is an  image. With **bold** text, and even the T<sub>E</sub>X logo.

Note that inside attributes, and also metadata values, a backslash followed by a newline acts as a line joiner and removes the newline and the following whitespace before processing the attributes. This is convenient when defining long strings inside attributes. For example, if a figure has a long caption, we could write it as:

```
~ Figure { #myfigure; \
    caption: "Here is a really \
    long caption." }
...
~
```

To get a literal line break inside an attribute string, use the entity `&nl;` instead.

### 5.3.2. CSS formatting support

Attributes that are not special to Madoko (as described in Section 5.3.7), are passed as CSS style attributes in the HTML backend. For the LaTeX backend, Madoko processes many CSS formatting commands and translates them to appropriate LaTeX commands (and ignores all others). Currently formatting commands that are recognized across backends are:

- `display`: (block | inline | inline-block | none)
- `margin`: [auto | length | percentage]{1,4}
- `margin-left, margin-right, margin-bottom, margin-top`
- `padding`: [auto | length | percentage]{1,4}
- `padding-left, padding-right, padding-bottom, padding-top`

- width: *length* | *percentage* | auto | available | normal
- height: *length* | *percentage* | auto | available | normal
- min-width, max-width
- min-height, max-height
- vertical-align: (top | middle | bottom | baseline | sub | super | text-top | text-bottom | *length*)
- border: [*width*] [*style*] [*color*] (in any order)
- border-style: (none | solid | dotted | dashed | double | groove | ridge | inset | outset)
- border-left-style, border-right-style, border-top-style, border-bottom-style.
- border-width: *length*
- border-left-width, border-right-width, border-top-width, border-bottom-width.
- border-color: *color*
- border-left-color, border-right-color, border-top-color, border-bottom-color.
- border-radius: *length* [*length*]
- border-top-left-radius, border-top-right-radius, border-bottom-left-radius, border-bottom-right-radius.
- background-color: *color*.
- background-clip: border-box | padding-box | content-box
- color: *color*
- text-align: (center | right | left | justify)
- text-indent: *length*
- line-height: *length*
- font-style: (italic | oblique | normal)
- font-variant: (small-caps | all-small-caps | petite-caps | all-petite-caps | normal)
- font-weight: (bold | bolder | lighter | normal | *number*)
- font-size: (xx-small | x-small | small | medium | large | x-large | xx-large | *length* | *percentage*)
- font-family: (monospace | serif | sans-serif | cursive | fantasy | normal | *family*)
- float: (left | right). Limited support at this time in LaTeX.
- list-style-type: decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | lower-latin | upper-latin | disc | circle | dash | square | none
- page-break-before, page-break-after: always | avoid | left | right | recto | verso | auto | *penalty*. In PDF output, auto is interpreted as a good page break (`\goodbreak`) and *penalty* is passed to the LaTeX `\penalty` command.

The following attributes are supported on images only:

- zoom: *fraction* | *percentage*
- transform-scale: *fraction* | *percentage*

- **transform-rotate**: *angle*. Anti-clockwise rotation in degrees.

Some attributes are only applied during PDF output:

- **breakable**: `true` | `false`. Makes the box breakable over a page.
- **height-align**: `top` | `middle` | `bottom`. Vertical alignment of the content in a fixed height box.
- **baseline**: `top` | `middle` | `bottom`. Placement of the baseline of a box.
- **page-align**: (`top` | `bottom` | `topbottom` | `page` | `here` | `forcehere` | `inplace`). Used for float placement in LaTeX for **Figure** custom blocks (see Section 4.13.1).

### 5.3.3. CSS font family

A font family is a comma separated list of fonts. In LaTeX, any font that starts with **tex-** is processed before any other fonts. There are the following ways to specify the font:

1. **serif**, **sans-serif**, **monospace**, **cursive**, **fantasy**: The standard CSS fonts.
2. **tex-family**-[*encoding*]/*family*[/*series*]/[*shape*]/[*scale*]: Specify an exact LaTeX font to use.
  - *encoding*: Optional LaTeX font encoding, for example T1 or OT1.
  - *family*: The LaTeX font family code, for example **cmr** for Computer Modern. Some codes can be found at [Wikipedia](#).
  - *series*: Optional, one of **l** (light), **m** (medium,normal), **b** (bold), or **bx** (extra bold).
  - *shape*: Optional, one of **n** (normal), **it** (italic), **sl** (slanted), **sc** (small-caps), or **bf** (bold-face).
  - *scale*: Optional, fraction to scale the font, for example 0.81.

For example, to typeset code fragments in HTML with *Consolas* and in PDF output using the *beramono* fonts (=fvm), you can specify a metadata rule:

```
pre,code {
  font-family: Consolas, "tex-family-T1/fvm/0.81";
}
```

3. **tex-cmd-cmd**: Specify a LaTeX command to use, for example **tex-cmd-sffamily**.
4. **tex-font**[*font options*]: Specify a general system font to be used in LaTeX only. You can pass the **\fontspec** font options between square brackets. For example: **font-family: "tex-Consolas[Scale=2,PunctuationSpace=3]"**.

5. *fontname*: In all other cases the name is directly passed to the `\fontspec` command. If using the default `xelatex` any system font can be used this way.




```
~ { font-family: Georgia }
This is in the Georgia font.
~
```

---

This is in the Georgia font.

#### 5.3.4. CSS colors

A *color* can be standard named color or use any of the CSS color formats, namely:

- `#rrggbb` (or `#rgb`) where `xx` is hexadecimal, e.g. `#0000FF` 
- `rgb(r,g,b)` where each component is a number between 0 and 255, or a percentage, e.g. `rgb(50%,50%,100%)` 
- `hsl(h,s,l)` where `h` is an angle between 0 and 360, and `s` and `l` percentages. Use 50% for the default lightness, e.g. `hsl(240,100%,80%)` 

The recognized named colors are any of the standard [CSS Level 4 colors](#). For example, the basic CSS colors are shown in Figure 6.

If you would like to use another named color that is not part of the CSS named colors, you can often just define an entity in the metadata:

`MyIndigo: #4B0088`

and use Madoko's expansion to use it inside an attribute:

```
This is "[MyIndigo]{color: &MyIndigo;}".
```

---

This is "MyIndigo".

#### 5.3.5. Complex CSS Layout

In general, it is hard to emulate a complex CSS layout in LaTeX so some particular combinations may not work as expected in LaTeX. We strive to make it work seamlessly though so please [report any issues](#). Nevertheless, the above

Name	Hex value	Color
Red	#FF0000	
Lime	#00FF00	
Blue	#0000FF	
Yellow	#FFFF00	
Cyan,Aqua	#00FFFF	
Magenta,Fuchsia	#FF00FF	
Maroon	#800000	
Green	#008000	
Navy	#000080	
Olive	#808000	
Teal	#008080	
Purple	#800080	
Orange	#FFA500	
Black	#000000	
DimGray	#696969	
Gray	#808080	
DarkGray	#A9A9A9	
Silver	#C0C0C0	
LightGray	#D3D3D3	
White	#FFFFFF	
Gainsboro	#DCDCDC	
FloralWhite	#FFFAF0	
Ivory	#FFFFF0	

---

**Figure 6.** The basic CSS colors



formatting commands should suffice in most situations and we can program already many fancy examples:

```
[**Aaaa[a]{vertical-align:-0.3ex}
[aa]{vertical-align:-0.7ex}
[r]{vertical-align:-1.4ex}
[g]{vertical-align:-2.5ex}
[h]{vertical-align:-5ex}**]{font-size:large; height:0pt }
he shouted but not even the next
one in line noticed that something
terrible had happened to him.
{ width: 18em; border: solid 1px black; \
  padding: 1ex; background-color: FloralWhite }
```

Aaaaa aa r he shouted but not  
even the next one in line noticed that some-  
thing terrible had happened to him.

The following example shows a complex box layout with background clipping and elliptical borders:

```
Here is the content of a complicated box with back&shy;ground clipping
and elliptical borders with smooth transitions.
{ margin-right:6em; \
  padding:1.5em; \
  background-color:floralwhite; \
  background-clip: padding-box; \
  border: 0.75ex solid black; \
  border-radius: 4ex; \
  border-top-left-radius: 8ex; \
  border-left-width: 1.5ex; \
  border-left-color: teal; \
  border-right-style: double; \
  width: 22.2em;\
}
```

Here is the content of a complicated box with back-  
ground clipping and elliptical borders with smooth  
transitions.

### 5.3.6. Floating blocks

There is limited support for the `float` attribute but it is generally quite hard to emulate this well in LaTeX. However, one can use the `float` attribute to put figures or general custom blocks on the left- or right-side of a page and have text flow around it. For example:

```
~ Figure { caption:"A formula"; width:60%; float:left; margin-right:1em }
~~ Math
e = mc^2
~~
~
"Aarghhh", he shouted but not even the next
one in line noticed that something
terrible had happened to him.
"Aarghhh", he shouted but not even the next
one in line noticed that something
terrible had happened to [him.]{float:right}
```

$$e = mc^2$$

**Figure 7.** A formula

happened to

“Aarghhh”, he shouted but not even the next one in line noticed that something terrible had happened to him. “Aarghhh”, he shouted but not even the next one in line noticed that something terrible had him.

Note that it is important to give a specific width when specifying `float`. Also, in LaTeX, the height of the text flowing around the float is not always correct (especially if the text contains mathematics). To fine-tune the appearance in LaTeX, one can use the `tex-wrap-lines` key to give the height of the float figure in text lines. Internally, Madoko uses the `wrapfigure` environment to render this. You can customize this with the `tex-wrap-env` key. When using wrapping, you can also specify the key `tex-float`: with the values `inside` or `outside` to wrap to the inside or outside of a page.

Still the LaTeX command is fragile and generally works best when immediately followed by a paragraph. Also, when using `float` on inline elements, it is implemented in LaTeX simply using `\hfill` which works for most common situations.

### 5.3.7. Special attributes

Normally, attributes are always passed to LaTeX as is, and in the HTML passed as CSS attributes. However, you can use special prefixes to have them end up at other places:

- **html-name**: use *name* as an HTML attribute only in the HTML backend. For example, you can use **html-target=\_top** for links inside frames.
- **css-name**: use *name* as a CSS attribute only in the HTML backend. For example, you can use **css-font-family=Cambria** for setting a font family only in the web page.
- **tex-name**: use *name* as an attribute only in the LaTeX backend.
- **data-name**: in the HTML backend uses a HTML attribute **data-name** instead of a CSS style attribute. Passed to LaTeX as is.

Some attributes have a special meaning to Madoko. In particular:

- **tight**: If **true** suppresses enclosing the first text content in this block as a paragraph. This is used for example for a **BibItem** block.
- **tag**: If **false** suppresses the output of a **div** tag (or LaTeX environment) for a custom block.
- **toc**, **toc-line**, **toc-depth**: Used for generating custom table of contents. Used for example by headings. See Section 4.11 for more information.
- **before**, **replace**, and **after**: Used to transform the content of the block. See Section 5.6.
- **start:num**: Gives the start number for an ordered list (Section 4.4).
- **target:frame**: Gives the HTML frame target for a link.
- **math-needpdf**: If **true**, forces the pdf conversion method for an equation (instead of using dvi). This is needed when using fancy LaTeX math using TikZ for example.
- **snippet-needpdf**: If **false**, uses the dvi conversion method for a snippet (instead of using pdf). This can improve performance of image extraction. (Section 4.10.6).
- **html-elem=elem**: Use *elem* as the HTML element instead of **div** or **span** for block and inline elements respectively.
- **id:id**: Sets the *id* of an element. Usually this is done using a hash name directly as **#id**.
- **class:classes**: Adds to the classes of this element. Usually set using a dot name as in **.class**. If you use **class=clear**, the classes are cleared. Similarly, you can use **.class=clear** to remove one particular class.
- **label:label**: Sets the label of an element. This is used when referencing this block as explained in Section 4.2
- **caption:caption**: Sets the caption of figure as explained in Section 4.3.
- **breakable:(true|false)**: Specifies if a block element can be broken across multiple pages. For example, for tables the LaTeX backend will use the **longtable** environment instead of the regular **tabular** environment (which cannot break across pages).

- `cite-label:label`: Sets the citation label of an `BibItem` block. This is used when referencing a bibliography item as explained in Section 4.12
- `cite-style:style`: Sets the citation style for a citation (Section 4.12.2).
- `tex-wrap-lines:lines`: Set the height of a float figure to *lines* height (Section 5.3.6).
- `sticky`: The attributes for this element will stick to apply to all of the same elements that follow in the document.
- `-:` clear any attributes that have been set before (usually through meta-data rules).
- `input:input`. Sets the input mode of this block which determines how the content of this block is processed. One of:
  - `pre`: Preformatted code (Section 4.8).
  - `raw`: Raw unprocessed input that is passed directly to the output.
  - `texraw`: Raw TeX code that is passed directly to LaTeX (See `TeXRaw`)
  - `htmlraw`: Raw HTML code that is passed directly to the HTML output (See `HtmlRaw`)
  - `math`: LaTeX mathematics mode input. This input is usually invoked through an `Equation` block as described in Section 4.10.
  - `mathpre`: Preformatted LaTeX mathematics. This input is usually invoked through a `MathPre` block (Section 4.10.7).
  - `mathdefs`: LaTeX math definitions. This input is usually invoked through a `MathDefs` block (Section 4.10.3).
  - `normal, markdown`: Regular Madoko markdown input (default).
  - `texonly`: Regular Madoko markdown that is only processed when generating LaTeX output. Usually used through the `TexOnly` block (Section 4.13).
  - `htmlonly`: Regular Madoko markdown that is only processed when generating HTML output. Usually used through the `HtmlOnly` block (Section 4.13).

See also the special attributes used for custom LaTeX styling in Section A.7.

### 5.3.8. Special attribute classes

Some class attributes are treated specially by Madoko. In particular:

- `.preview`. This class is set on the `<body>` element in the preview and can be used to style elements differently in the preview versus the full document.
- `.para-block`. If a block element has the `.para-block` class, it will be considered part of the preceding paragraph (and not end that paragraph). This is for example done for the `Equation` block since equations are part of a paragraph. This improves typography significantly, especially for the LaTeX backend. Madoko will also assign the `para-continue` class to the paragraph preceding a `.para-block` element.
- `.para-end`: Normally, equations have a `.para-block` class and are thus

considered part of the paragraph. If you would like such block to be the end of a paragraph, you should assign the `.para-end` class to end it explicitly. In the LaTeX backend this will cause the next paragraph to be indented.

- `.indent`. This class is automatically added by Madoko to any paragraph that follows other paragraphs or `.para-blocks` (and is not inside a list). This can be used to switch from *block* mode paragraphs to *indented* paragraphs using some CSS rules: `p.indent { text-indent: 1em }`
- `.align-center`, `.align-right`, `.align-left`. These classes align the *content* of a block (instead of the block itself) to the center, right, or left.
- `.hidden`. Hides the content of the block.
- `.block`. Treat this as a block element instead of inline content. For example, in HTML output this will add a top and bottom margin to the element.
- `.wide`. Used for **Figure** blocks: in a two-column output, this figure will span both columns horizontally (see also Section 4.13.1).
- `.free`, `.textual`. These classes are used for typesetting citations (see Section 4.12.2).
- `.list-star`, `.list-dash`, `.list-plus`. Used to customize appearance of bullets in a list.
- `.code`, `.coden`: added to inline code elements where *n* is the number of back-quotes. This way you can add special styling to double quote inline code for example.
- `.pre-fenced`, `.pre-fencedn`: Added to fenced code blocks where *n* is the number back-quotes.
- `.pre-indented`: Added to indented code blocks.
- `.math-inline`: Added to inline math fragments.
- `.math-display`: Added to block math fragments.

## 5.4. Metadata rules

Often, we need to apply specific attributes and formatting to certain elements or custom blocks. This can be done using *metadata rules*. These rules function much like regular CSS rules except that they are quite a bit more limited. You can use a chain of an element name, identifier (`#id`), or class names (`.class`) for matching. The value of a rule are attributes that get applied to any matching block. Here are some examples:

```
Blockquote { font-style: oblique }
.bold      { font-weight: bold }
#myblock   { border: 1px solid black }
```

This would typeset a block quotes in an oblique font, any block with a **bold** class in bold, and the block with the `myblock` id with a solid border. You can also use a comma separated list of matches that apply to all. For example, we can give all code, inline or as a block, the class `prettyprint` as:

```
pre,code { .prettyprint }
p.indent, blockquote.indent { text-indent: 1em }
```

### 5.4.1. Names of predefined elements

Most standard Madoko block elements, like lists or block quotes, can also be targeted by rules. Their element names are the standard HTML names, namely:

Element	Name
Paragraph	<code>p</code>
Code block	<code>pre</code>
Code inline	<code>code</code>
Code escaped text	<code>code-escaped</code>
Block quote	<code>blockquote</code>
List	<code>ul</code>
Numbered list	<code>ol</code>
List item	<code>li</code>
Definition list	<code>dl</code>
Definition term	<code>dt</code>
Definition	<code>dd</code>
Horizontal rule	<code>hr</code>
Table	<code>table</code>
Heading	<code>h1,...,h6</code>
Unnamed custom block	<code>div</code>
Named custom block	<i>name</i>

Note that every custom block *blockname* automatically also gets the class *blockname* and is thus matched by both *blockname* and *.blockname* rules. This can be also be useful when applying styles in CSS rules.

Some standard elements also get standard class names assigned to make it easier to target them with specific rules:

Element	Class name
Code block that was indented	<code>.pre-indented</code>
Code block that was fenced	<code>.pre-fenced</code>
Code block with <i>n</i> back-quotes	<code>.pre-fenced<i>n</i></code>
Code inline with <i>n</i> back-quotes	<code>.coden</code>
Math inline	<code>.math-inline</code>
Math block	<code>.math-display</code>
List using *	<code>.list-star</code>
List using +	<code>.list-plus</code>
List using -	<code>.list-dash</code>

For example, you could specify that any indented code blocks should use the javascript syntax highlighter:

```
.pre-indented { language:javascript }
```

Or typeset double-quoted inline code (i.e. ```code```) with a gray background:

```
.code2 { background-color: gainsboro }
```

### 5.4.2. Advanced: Styling in CSS

Sometimes we only want to apply certain styling to just HTML. For HTML, styling through classes works best: we can simply write some CSS (and include it through the `Css` or `Css Header` metadata or directly within `<style>` tags.). For example,

```
<style>
.slanted { font-style: oblique; font-family: Cambria }
</style>
```

Note that any custom block in Madoko (like `~ Slanted`) automatically gets assigned the class name `slanted` and can thus be matched easily in CSS.

For styling in LaTeX, or when certain CSS keys are not yet processed in LaTeX, see Appendix [A.7](#).

## 5.5. Numbering

For larger documents, numbering sections, figures, tables, images, etc. is quite important and Madoko supports this well. By default, Madoko will number headers, figures, and equations. Numbering for headers is by default up to 3 levels, but it can be set for the document using the `Heading depth` meta variable. If it is set to zero, it suppresses numbering for headings completely:

Heading Depth: 0

### 5.5.1. Advanced: Custom numbering

The numbering can be completely customized though. Counters are introduced using `@name` syntax in attributes. When such counter name occurs, it is automatically incremented. If the value of a *label* contains a counter name, it is replaced by its current value. For example, the default attributes for equations contain:

```
{ @equation; label:"(@equation)" }
```

This automatically will increment the `@equation` counter at each occurrence of an equation block element, and set its label to the current value surrounded by parenthesis. Similarly, the default label for main sections is defined as:

```
{ @h1; label:"@h1" }
```

Actually, Madoko automatically increments counters with the same name as the block, so in the previous two cases, we should leave out `@equation` and `@h1` or we do double increments.

Of course, for headers we like to display the label by default in front of the header text. This can be done using the `before` attribute where we use the `&label;` key to insert the value of the label. So, the full definition for level 1 headings is:

```
{ label:"@h1"; before:"&label;.&ensp;" }
```

### 5.5.2. Advanced: Reset counters

If a counter has a dash in its name, of the form `@prefix-name`, then the counter will be reset on every increment of the counter *prefix*. For example, the counter for subsections is reset on every new section (using [metadata rules](#)):

```
h2 {
  @h1-h2; label:"@h1.@h1-h2"; before:"&label;.&ensp;";
}
```

Similarly, a counter like `@h1-h2-h3` would reset on every increment of counter `@h1` or `@h1-h2`.

### 5.5.3. Advanced: Display format

The counters above are displayed as arabic numbers but sometimes we would like to display a number using other formats. When you assign a single letter to a counter, it will continue counting using letters. This is nice for example when starting the appendix where each section is generally numbered using letters:

```
# Appendix (not numbered at all)  {-}

# Section one of the appendix { @h1:"A" }
This section will be numbered as 'A'

# Another appendix section
This section will be numbered as 'B'
```

The display format of a counter can also be set independently of setting its value. Currently, Madoko supports `arabic/decimal`, `arabic0/decimal0`, `upper-case/upper-alpha/upper-latin`, `lower-case/lower-alpha/lower-latin`, `decimal-leading-zero`, `lower-roman`, `upper-roman`, `lower-greek`, `upper-greek`, `cjk-decimal`, `symbolic`, `circled-decimal`, `disc`, `square`, `circle`, `dash`, and `none`. The style `arabic0` starts counting at 0. Assigning a single letter, like `@h1:"A"` in the previous example is just a shorthand for:

```
{ @h1:upper-alpha; @h1:1 }
```

As a final example, suppose we would like to count figures using lower-case letters and per section. We can do this by defining the default attributes for figures as a metadata rule:

```
Figure {
```



```
@h1-figure:lower-alpha; @h1-figure; label:"@h1\/-@h1-figure";
}
```

The label definition here displays figure labels of the form 2-b for example (where we used the empty [escape sequence](#) `\/` to prevent the counter name `@h1` being read as `@h1-`). Note that because this is a metadata rule, we could not use the assignment `@h1-figure:"a"` here, or otherwise every figure would get numbered as `a`. In this case we just want to set the display mode here and not a specific value. Also note that we still need to explicitly increment the counter too (using a plain `@h1-figure`) since just setting the display format does not increment the counter.

## 5.6. Advanced: Replacement

Madoko has three attributes that can transform the content of block or inline element, namely `before`, `after`, and `replace`. The `before` and `after` elements just add content before and after:

```
~ Myblock { before="*Myblock*: " after="." }
the content
~
```

---

*Myblock*: the content.

The `replace` attribute replaces the entire content with its value. You can have multiple replacers and they are all applied in order. The replacers can be cleared using the special `clear` value. Finally, all replacers can contain entity names (Section 5.2) which are expanded. Useful entities are `&source;` which expands to the current content of the block (possibly having already some replacers applied), and `&nl;` which expands to a newline character. In particular, `before` and `after` are defined in terms of `replace`, where `before=value` is just syntactic sugar for `replace=value\/&source;` and similarly for `after`.

As an advanced example, for this document I defined a metadata rule for the `Sample` custom block that replaces its content by both a code block and a regular markdown block. In a simplified form, it is defined as:

```
Sample {
  replace:"~ Begin SampleBlock&nl;\n
    ~~~~&nl;&source;&nl;~~~~&nl;\n
    ---- &nl;&source;&nl;\n
    ~ End SampleBlock"
}
```

### 5.6.1. Advanced: Regular expression replacement

A **replace** attribute can also define a general [regular expression](#) replacement of the form `/regex/replacer/(g|i|m|c)?`. The regular expression *regex* is matched against the content, and a match is replaced by *replacer*. The options are:

- **g**: instead of just the first match, it will globally replace all matches found in the content.
- **i**: use case-insensitive matching.
- **m**: do a multi-line match where `^` and `$` match the beginning and end of each line respectively instead of just the start and end of the content.
- **c**: do case conversion; enlarges valid escape characters with one of `luLUE` which can be used to do case conversion, see the next section for more information.

Inside the *replacer* we can use the following escape sequences:

- `\digit`: gets replaced by the *digit* capture group in *regex*.
- `\/`: becomes a forward slash.
- `\\`: becomes a single backward slash.

For example, here is an example where we replace `<quoted>` text by single guillemet quotes:

```
~ { replace="/<(.*?)>/&lrsquo;\1&rsquo;/g" }
Here is < quoted > text.
~
_____
Here is ‹ quoted › text.
```

#### Case conversion

When we pass the `c` flag, we can also use special case conversion escape sequences in the *replacer* expression:

- `\U` or `\L` transform the following text up to the next `\E` (or the end of the replacement) to upper- or lower-case respectively.
- `\u` and `\l` replace the following character to upper- or lower-case.

```
~ { replace="/(\w+)/\u\1/gc" }
all words to title-case.
~
_____
All Words To Title-Case.
```

### Mapping strings

The `replace` attribute can also perform a *mapping* when it has the form:

```
// regex1 / repl1 // ... // regexn / repln //(g|i|m|c).
```

A mapping matches against the regular expression  $(regex_1)|\dots|(regex_n)$ , and when it finds a match on some group  $i$ , it applies the replacer  $repl_i$ . The replacement expression can use capture groups as usual.

For example, we could device a *greek* or *hiragana* mode replacement. Here is a simplified example:

```
~ Greek { replace:"//a/&alpha;//d/&delta;//n/&nu;//g" }
greek daan.
~
~ Hiragana {replace:"//ko/&#12371;//ni/&#12395;\
                //\\(.)\\1//chi/&#12385;//ha/&#12399;\
                //n/&#12435;\
                //g"; \
                font-family: "MS Gothic"; }
konnichiha is Japa\nese.
~
_____
greek δααν.
こんにちは is Japanese.
```

Note that the previous example generally needs a `font-family` specification to display correctly in LaTeX since Japanese characters are not in the standard font. See Appendix A.8 for more information.

#### 5.6.2. Advanced recursion and replacement

As more advanced example of recursion and replacement, we can actually calculate [fibonacci numbers](#). In the following sample,  $n$  number of `x` characters are replaced by  $fib(n)$  `y` characters:

```
How many `y`s is the Fibonacci of 5 `x`s?
~ Fib
xxxxx
~
_____
How many y's is the Fibonacci of 5 x's?yyyyyyyy
```

This is done through the following metadata rule:

```
Fib {  
  replace: '/^x?$/y/';  
  replace: '/xx(x*)/~Fib&nl;x\1&nl;~&nl;~Fib&nl;\1&nl;~/';  
  tag:false;  
  tight:true;  
}
```

The first replacer will replace a single optional `x` with a `y`. The second one matches 2 or more `x`'s, and replaces these recursively by two new `Fib` blocks: one with  $n-1$  and one with  $n-2$  `x` characters. These blocks will get processed now recursively. Finally, by using the `tag:false` attribute we suppress the inclusion of many `div` elements in the HTML, while the `tight` attribute suppresses the addition of paragraph elements.

This example shows the replacement facility of Madoko is quite powerful. It is even possible to define a generic [SKI combinator expander](#) which makes Madoko's replacement mechanism (almost) Turing complete<sup>5</sup>.

---

<sup>5</sup>Almost, since Madoko expands only up to a certain limit and since regular expressions cannot express arbitrary nesting levels.

# References

- [1] J. Fagerberg, D.C. Mowery, and R.R. Nelson, editors. *Oxford Handbook of Innovation*. Volume 1. Oxford University Press, Oxford. 2004.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley, Reading, Massachusetts. 1993.
- [3] O. Grandstrand. “Innovation and Intellectual Property Rights,” in Fagerberg et al. [1], volume 1, chapter 10. 2004.
- [4] Donald E. Knuth. *The T<sub>E</sub>X Book*. Addison-Wesley. 1984.
- [5] Leslie Lamport. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System (2nd Edition)*. Addison-Wesley. 1994. See also [4].



## Chapter 6

# Appendix

### A.1. Command line options

Generally Madoko is invoked on the command line as:

- `madoko [options] files`

where the options can consist of:

Short	Long	Description
	<code>--version</code>	Display version information.
<code>-v</code>	<code>--verbose</code>	Be more verbose.
	<code>--odir=dir</code>	Write output files to the specified directory (=out)
	<code>--xmp</code>	Only process markdown between <code>&lt;xmp&gt;</code> tags.
	<code>--tex</code>	Output a LaTeX file too.
	<code>--pdf</code>	Generate a PDF file (implies <code>--tex</code> ).
	<code>--texzip</code>	Generate TeX zip for submission
	<code>--png</code>	Use PNG instead of SVG for math in HTML
	<code>--convert-tex</code>	Convert input from TeX to Markdown
<code>-f</code>	<code>--fragment</code>	Generate a fragment instead of a full document.
	<code>--logo</code>	Add a ‘Created with Madoko.net’ message.
	<code>--sandbox</code>	Run in a sandbox for secure server execution.
	<code>--sanitize</code>	Always escape or suppress user defined html.
	<code>--pedantic</code>	Pedantic mode.
	<code>--bench</code>	For benchmarking: turn off numbering, etc.
	<code>--installdir=dir</code>	Set installation directory explicitly.
<code>-r</code>	<code>--rebuild</code>	Force rebuild bibliography, math, etc.
	<code>--prelude=file</code>	Include <i>file</i> at start of the document
<code>-mkey:val</code>	<code>--meta=key:val</code>	Semi-colon separated list of metadata values

All the boolean flags can be negated by prefixing them with `no-`, for example to suppress the logo message at the end of a document, use `--no-logo`.

The verbose flag (`-v`) can be repeated to become more verbose. For example,

by using `-vv` LaTeX warnings and the size of math images are displayed.

Normally, Madoko writes the output files to the `out` directory (relative to the input file) but you can override this with the `--odir` option. The HTML output for a file `file.mdk` consists of `file.html` and `madoko.css`. The latter contains standard styling for Madoko and is necessary for example to display lines in tables correctly. For LaTeX output (`--tex`), Madoko also emits `file.tex` and the `madoko2.sty` style file which contain necessary commands to process the Madoko output in LaTeX.

The `--texzip` option lets you generate a zip file that contains all necessary files to generate a PDF using LaTeX. Often this is required for academic submissions. Madoko will automatically include any necessary images, packages, etc.

The `--xmp` option instructs Madoko to only process content between `<xmp>` and `</xmp>` tags. This can be useful if the main body of your document is in HTML and only some parts are written in Madoko.

The `--fragment` flag forces Madoko to generate a document fragment. By default, a *full* document is generated which is self sufficient and contains for example for HTML proper `<head>` and `<body>` elements instead of just the markup for the content. Similarly, for LaTeX, a full document contains an `\documentclass`, `\begin{document}` and necessary `\usepackage` commands. For a fragment, just the content is generated without those headers and footers.

The `-m` flag can be used to set metadata values. For example, when specifying what command to use to invoke LaTeX, we can say `-mpdflatex:/foo/bar/xelatex` for example.

The `--sandbox` flag runs Madoko in a sandbox: it can only read and write files underneath the document directory (actually, Madoko will still read a predefined set of styles from the `styles` directory, like `prelude.mdk` and `madoko2.sty`). Moreover, documents cannot change the following metadata keys: `latex`, `pdflatex`, `dvisvg` `dvipng`, `math-latex`, `math-latex-full`, `bibtex`, `math-convert`, `convert`, `ps2pdf`, and `dvips` (actually, you can still set the `latex` keys to a predefined set of safe latex variants, i.e. `latex`, `pdflatex`, and `xelatex`). Note that this flag only influences the Madoko program, any run of LaTeX or other external programs needs to be secured separately. In particular, on a server we recommend using `TexLive` where Madoko will automatically pass an environment with `openany_in` and `openany_out` set to `paranoid`, disabling parsing of the first line of input, and disabling any shell command execution.

## A.2. Slide shows and presentations

Madoko can generate excellent slide shows using either the [reveal.js](#) library in HTML or the [Beamer](#) package in LaTeX.

Start a slide show with including the `Presentation` style at the start of the document:

```
[INCLUDE=presentation]
```



Slides are now automatically started on a level 1 (#) or 2 (##) header, or by using a `~Slide` custom block explicitly. Use the `.fragment` class on any element to create overlays that appear one by one. The `.fragmented` class can be used on a list to have each item appear in order. For example:

```
# my first slide
```

```
And some content
```

```
And more
```

```
{.fragment}
```

```
And more
```

```
{.fragment}
```

```
# and my second slide
```

```
* with
```

```
* a list
```

```
* in it
```

```
{.fragmented}
```

The HTML slide shows support the `~Notes` custom block for speaker notes, and the `~Vertical` custom block to create a set of vertical slides.

Here is a small [example slide show](#) written in Madoko ([html](#), [pdf](#), [source](#)).

### A.2.1. Using Reveal.js

The HTML backend works with the [reveal.js](#) library:

- It uses the [reveal.js library](#) from a common CDN url. You can set your own using the `Reveal Url` metadata:

```
Reveal Url: https://cdn.jsdelivr.net/reveal.js/2.6.2
```

- Select another default theme by setting the `Reveal Theme`, for example `Reveal Theme: sky`.
- Use the `data-background` attribute on headings to set the background color or background image for a slide.
- Use the `data-transition` attribute on headings to set the transition animation. For example:

```
# Slide { data-transition:zoom }
## Next slide
```

- The Javascript `revealConfig` variable contains the default configuration options and can be extended inside `<script>` tags.

### A.2.2. Using Beamer

The LaTeX backend uses the [Beamer](#) package:

- Set the `Beamer Theme` metadata key to select another theme, like `singapore` for example.
- Set the `Beamer Theme Options` for different options for the theme.
- Use the `.pause` class to insert pauses anywhere in a slide.

## A.3. Advanced: Customizing citations

For each citation, you can actually change how it is displayed by setting the `cite-style` attribute on the citations:

First a natural citation `[@Goo93]{cite-style:natural}`,  
then a super one `[@Goo93]{cite-style:super}`.

First a natural citation (Goossens et al., 1993), then a super one<sup>2</sup>.

Moreover, we can customize the formatting of citations by specifying braces etc. The general format of a citation style is:

*base* `[: (sort|nosort)] [: open, close, sep [, aysep] [, yysep]]`

where *base* is one of `natural`, `numeric`, `textual`, or `super`, and the values *open* to *yysep* are double quoted strings:

- *open*: the opening brace.
- *close*: the closing brace.
- *sep*: separator between citations.
- *aysep*: Optional separator between authors and years. Equal to *sep* by default.
- *yysep*: Optional separator between years with a common author. Equal to *aysep* by default.

For example:

Switch to a compact bold numeric style  
`[@Goo93;@Fberg04]{cite-style:'numeric:["**","**"],"**;**"}'`  
or an unsorted super  
style `[@Lamport:LaTeX;@Knuth:TeX]{cite-style:super:nosort}`

Switch to a compact bold numeric style [\[1;2\]](#) or an unsorted super style<sup>5,4</sup>

Clearly, one should usually only do this for the `Cite Style` metadata key, and not sprinkle this kind of formatting through your prose.

## A.4. Advanced: Not using BibTeX

If necessary, it is possible to completely circumvent using BibTeX and write your bibliography entries by hand. First, we need to ensure Madoko will not run the BibTeX tool for us:

```
BibTeX: False
```

Next, we write our bibliography entries inside a `Bibliography` block:

```
~ Bibliography { caption:"00" }
...
~
```

where the `caption` is only used in LaTeX output and should be a string that is the widest label necessary for the numeric style. Inside the bibliography block, you can put `Bibitem` blocks for each bibliography entry. For example:

```
~~ Bibitem { #WadlerThiemann03 }
Philip Wadler and Peter Thiemann.
The marriage of effects and monads.
ACM Trans. Comput. Logic, 4(1):1--32, 2003.
~~
```

Moreover, you can add a `caption` that is displayed when hovering over a citation, and a `searchterm` that is used when the user clicks on the magnifying glass icon:

```
~~ Bibitem { #WadlerThiemann03; \
  caption:"Wadler and Thiemann: \
    The marriage of effects and monads" \
  searchterm="Wadler+Thiemann+Marriage+Effects+Monads" }
Philip Wadler and Peter Thiemann.
The marriage of effects and monads.
ACM Trans. Comput. Logic, 4(1):1--32, 2003.
~~
```

The above example is for numeric style citations. For author-year citations, the entry should have an explicit `cite-label`:

```
~~ Bibitem { #remy; cite-label:'R  my(1993)' }
Didier R  my.
Type inference for records in a natural extension of ML.
In Carl   A. Gunter and John   C. Mitchell, editors,
Theoretical Aspects Of Object-Oriented Programming.
Types, Semantics and Language Design. MIT Press, 1993.
~~
```

```

~~ Bibitem { id:'nielson:polyeffect'; \
cite-label:'Nie:lson et\ al.(1997)Nielson, Nielson, and Amtoft'}
Hanne\ Riis Nielson, Flemming Nielson, and Torben Amtoft.
_Polymorphic subtyping for effect analysis:
The static semantics._
In Selected papers from the 5th LOMAPS Workshop on
Analysis and Verification of Multiple-Agent Languages,
pages 141--171, 1997. ISBN 3-540-62503-8.
~~

```

A `cite-label` should have the form *authors(year)longauthors*, where the *longauthors* are optional. Madoko parses these labels to correctly format citations.

## A.5. Advanced: packages in dynamic math mode

In dynamic math mode packages are not loaded since MathJax cannot handle arbitrary LaTeX packages. We need a special MathJax ‘extension’. For example, the `amscd` package is available as `AMScd` and we can use it in MathJax as:

MathJax Ext: `AMScd`

Another package that is available in both Latex and MathJax is the `mhchem` package:

MathJax Ext: `mhchem`

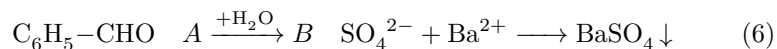
Package : `mhchem`

which can be used to easily draw chemical formulas:

```

~ Equation { #eq-chem; caption:"A chemical equation" }
\ce{C6H5-CHO}\quad
\ce{$A$ ->[\ce{+H2O}] $B$}\quad
\ce{SO4^2- + Ba^2+ -> BaSO4 v}
~

```



In the previous examples, it may seem superfluous to specify both a Math-JaX extension and LaTeX package but not in all cases the names or package functionality happens to be exactly the same.

## A.6. Advanced: Using Prettify to highlight code

If you are not fond of static highlighting, you can also highlight syntax dynamically using [Google Prettify](#). Of course, this only works for HTML output, PDF output is still highlighted using Madoko. To enable prettify, first include the Google script in the metadata (see Section 5.1):

Script: `https://google-code-prettify.googlecode.com/  
svn/loader/run_prettify.js { .preview}`

and add the `.prettyprint` class using attributes:

```

<code>... javascript { .prettyprint; .linenums }
function hi() {
  return "hi";
}
...

function hi() {
  return "hi";
}
  
```

Note that Madoko automatically disables static syntax highlighting in the HTML output if the `.prettyprint` class is present.

## A.7. Advanced styling in LaTeX

Sometimes we need to style LaTeX code specially, or handle cases where the CSS elements are not yet supported by Madoko (as described in Section 5.3.2). The following attributes can be used to customize the LaTeX output:

- **tex-cmd:***cmd*: The *cmd* is literally applied to the (braced) content of the element. For example: `[italic in tex]{tex-cmd: "\textit"}`. If you start the *cmd* with a left brace (`{`) the command is inserted before the content and braced on the outside. For example, `[italic using a switch]{tex-cmd: "{\itshape}"}`.
- **tex-env:** *env*[*args*]: Enclose the content of the element between a `\begin{env}args` and `\end{env}`. The *args* arguments are optional. For example, for an unordered list, the prelude defines the rule
 

```
ul { tex-env: "itemize" }
```
- **tex-cmd-inner**, **tex-cmd-outer**, **tex-env-inner**, **tex-env-outer**: The **tex-cmd** and **tex-env** are applied after ‘block css’, i.e. the margin, border, and padding, but before any ‘inline css’, i.e. fonts, indentation, etc. The

`inner` variants apply the command last after all other formatting has been done, while the `outer` variant applies before any other formatting. For example, the prelude defines the environment for figures as:

```
figure { tex-env-outer: figure[&tex-float-placement;] }
```

- `tex-env-postfix`, `tex-env-outer-postfix`, `tex-env-inner-postfix`, `tex-cmd-postfix`, `tex-cmd-outer-postfix`, `tex-cmd-inner-postfix`: Literally append the value to the environment or command. For example, the prelude uses:

```
figure.wide { tex-env-outer-postfix: "*" }
```

- `tex-cmd-before`, `tex-cmd-after`, `tex-cmd-outer-before`, `tex-cmd-outer-after`, `tex-cmd-inner-before`, `tex-cmd-inner-after`: Insert a command right before or after, the most inner, middle, or most outer content. You can start with a left brace (`{`) to brace the command and content on the outside. For example, the prelude defines the following rule for list items:

```
li { tex-cmd-before: "\item" }
```

- `tex-tabcolsep:length`: On tables sets the table column separation.
- `tex-label-before`, `tex-label`: If present issues a `\label` command before or after the outer element content. Usually handled already by the prelude for any element with an identity.
- `tex-tooltip`: For links, issues a `\mdtooltip` command after the outer content – this will show a small tooltip in PDF output.

## A.8. Unicode characters

Madoko already recognizes all named html entities and translates them to the appropriate LaTeX commands (Appendix A.9). If you often need a specific unnamed unicode character, it can be convenient to define a shorthand for it in the metadata:

```
llb: &#10214;
```

```
rrb: &#10215;
```

and then use those entity names instead. For example:

```
&llb;hi&rrb;.
{font-family: "Segoe UI Symbol"}
```

---

[[hi]].

### A.8.1. Unicode in LaTeX

Unfortunately, in LaTeX these unicode characters might be unknown. You can define your own definition for it using the LaTeX `\mdDefineUnicode` command in a `Tex Header` metadata key:

`Tex Header`:

```
\mdDefineUnicode{10214}{\ensuremath{\llbracket}}
\mdDefineUnicode{10215}{\ensuremath{\rrbracket}}
```

If no definition is given, LaTeX will work but output the entity as is, i.e. `&#10214;` for `#llbracket`. Under a unicode aware LaTeX, like [XeLaTeX](#) or [LuaLaTeX](#), it will call `\mdUnicodeChar` which will use the `\char` command to directly select the glyph from the current font.

### A.8.2. Unicode font selection in LaTeX

The previous examples works fine if only some unicode characters are used, but becomes cumbersome if you need many unicode characters, for example, when writing Japanese or Chinese characters as shown in our previous example in [Section 5.6.1.2](#). In this case, we need to select a font family in LaTeX that supports the glyphs directly. For example, MS Gothic or SimSun:

```
~ { font-family:"MS Gothic"}
&#12371;&#12435;&#12395;&#12385;&#12399; is Japanese.
~
```

---

こんにちは is Japanese.

Instead of using explicit unicode characters, you can also directly use UTF8 text files with the characters shown directly. For example:

```
Directly in unicode: [ ]{ font-family:"MS Gothic"}
(probably not visible in LaTeX monospace fonts)
```

---

Directly in unicode: こんにちは (probably not visible in LaTeX monospace fonts)

If most of your document is using such characters, the `\setmainfont` command of the `fontspec` package can set the default for the entire document instead of using attributes. See the `fontspec` package [documentation](#) for more information.

## A.9. Recognized character entities

Madoko recognizes all HTML5 named character entities and translates them correctly in LaTeX. Besides the standard named entities, Madoko also recognizes some extra named entities, like *bar*, *bslash*, *pagebreak*, *strut*, etc. These are denoted in the table with a star. The full list of predefined entities is:

number	name	glyph	remark
34	quot	"	
35	hash*	#	
36	dollar*	\$	
37	perc*	%	
38	amp	&	
39	apos	'	
40	lpar*	(	
41	rpar*	)	
42	ast*	*	
43	plus*	+	
47	fslash*	/	
60	lt	<	
62	gt	>	
92	bslash*	\	
94	caret*	^	
95	underscore*	_	
96	grave*	`	
123	lcurly*	{	
124	bar*		
125	rcurly*	}	
126	tilde*	~	
160	nbsp		~ in LaTeX
161	iexcl	!	
162	cent	¢	
163	pound	£	
164	curren	¤	
165	yen	¥	
166	brvbar		
167	sect	§	
168	uml	¨	
169	copy	©	
170	ordf	ª	
171	laquo	«	
172	not	¬	
173	shy		
174	reg	®	
175	macr	¯	



---

176	deg	°
177	plusmn	±
178	sup2	<sup>2</sup>
179	sup3	<sup>3</sup>
180	acute	´
181	micro	μ
182	para	¶
183	middot	·
184	cedil	¸
185	sup1	<sup>1</sup>
186	ordm	º
187	raquo	»
188	frac14	<sup>1</sup> / <sub>4</sub>
189	frac12	<sup>1</sup> / <sub>2</sub>
190	frac34	<sup>3</sup> / <sub>4</sub>
191	quest	¿
192	Agrave	À
193	Aacute	Á
194	Acirc	Â
195	Atilde	Ã
196	Auml	Ä
197	Aring	Å
198	AElig	Æ
199	Ccedil	Ç
200	Egrave	È
201	Eacute	É
202	Ecirc	Ê
203	Euml	Ë
204	Igrave	Ì
205	Iacute	Í
206	Icirc	Î
207	Iuml	Ï
208	ETH	Ð
209	Ntilde	Ñ
210	Ograve	Ò
211	Oacute	Ó
212	Ocirc	Ô
213	Otilde	Õ
214	Ouml	Ö
215	times	×
216	Oslash	Ø
217	Ugrave	Ù
218	Uacute	Ú
219	Ucirc	Û

---

220	Uuml	Ü
221	Yacute	Ý
222	THORN	Þ
223	szlig	ß
224	agrave	à
225	aacute	á
226	acirc	â
227	atilde	ã
228	auml	ä
229	aring	å
230	aelig	æ
231	ccedil	ç
232	egrave	è
233	eacute	é
234	ecirc	ê
235	euml	ë
236	igrave	ì
237	iacute	í
238	icirc	î
239	iuml	ï
240	eth	ð
241	ntilde	ñ
242	ograve	ò
243	oacute	ó
244	ocirc	ô
245	otilde	õ
246	ouml	ö
247	divide	÷
248	oslash	ø
249	ugrave	ù
250	uacute	ú
251	ucirc	û
252	uuml	ü
253	yacute	ý
254	thorn	þ
255	yuml	ÿ
256	Amacron	Ā
257	amacron	ā
258	Abreve	Ă
259	abreve	ă
260	Aogonek	Ą
261	aogonek	ą
262	Cacute	Ć
263	cacute	ć
264	Ccirc	Ĉ

---

265	ccirc	ĉ
266	Cdota	Ć
267	cdota	ć
268	Ccaron	Č
269	ccaron	č
270	Dcaron	Ǧ
271	dcaron	ǧ
272	Dstroke	Đ
273	dstroke	đ
274	Emacron	Ē
275	emacron	ē
276	Ebreve	Ė
277	ebreve	ė
278	Edota	Ė
279	edota	ė
280	Eogonek	Ę
281	eogonek	ę
282	Ecaron	Ě
283	ecaron	ě
284	Gcirc	Ĝ
285	gcirc	ĝ
286	Gbreve	Ğ
287	gbreve	ğ
288	Gdota	Ġ
289	gdota	ġ
290	Gcedil	Ģ
291	gcedil	ģ
292	Hcirc	Ĥ
293	hcirc	ĥ
294	Hstroke	Ħ
295	hstroke	ħ
296	Itilde	Ĩ
297	itilde	ĩ
298	Imacron	Ī
299	imacron	ī
300	Ibreve	İ
301	ibreve	ı
302	Iogonek	Į
303	iogonek	į
304	Idota	İ
305	idotless	ı
306	IJ	IJ
307	ij	ij
308	Jcirc	Ĵ

---

309	jcirc	ĵ
310	Kcedil	Ḳ
311	kcedil	ḱ
313	Lacute	Ł
314	lacute	ł
315	Lcedil	Ḷ
316	lcedil	ḷ
317	Lcaron	Ľ
318	lcaron	ľ
321	Lstroke	Ł
322	lstroke	ł
323	Nacute	Ń
324	nacute	ń
325	Ncedil	Ṇ
326	ncedil	ṇ
327	Ncaron	Ñ
328	ncaron	ñ
329	napos	'n
330	Neng	Ṅ
331	neng	ṅ
332	Omacron	Ȫ
333	omacron	ȫ
334	Obreve	Ȭ
335	obreve	ȭ
336	Odacute	Ȯ
337	odacute	ȯ
338	OElig	Œ
339	oelig	œ
340	Racute	Ŕ
341	racute	ŕ
342	Rcedil	Ṛ
343	rcedil	ṛ
344	Rcaron	Ř
345	rcaron	ř
346	Sacute	Ŝ
347	sacute	ŝ
348	Scirc	Ŝ
349	scirc	ŝ
350	Scedil	Ș
351	scedil	ș
352	Scaron	Š
353	scaron	š
354	Tcedil	Ṫ
355	tcedil	ṭ

---

356	Tcaron	$\mathring{T}$	
357	tcaron	$\mathring{t}$	
358	Tstroke		
359	tstroke		
360	Utilde	$\tilde{U}$	
361	utilde	$\tilde{u}$	
362	Umacron	$\bar{U}$	
363	umacron	$\bar{u}$	
364	Ubreve	$\breve{U}$	
365	ubreve	$\breve{u}$	
366	Uring	$\mathring{U}$	
367	uring	$\mathring{u}$	
368	Udacute	$\acute{U}$	
369	udacute	$\acute{u}$	
370	Uogonek	$\mathring{U}$	
371	uogonek	$\mathring{u}$	
372	Wcirc	$\mathring{W}$	
373	wcirc	$\mathring{w}$	
374	Ycirc	$\mathring{Y}$	
375	ycirc	$\mathring{y}$	
376	Yuml	$\mathring{Y}$	
377	Zacute	$\acute{Z}$	
378	zacute	$\acute{z}$	
379	Zdota	$\mathring{Z}$	
380	zdota	$\mathring{z}$	
381	Zcaron	$\mathring{Z}$	
382	zcaron	$\mathring{z}$	
383	slong	$\mathfrak{f}$	
402	fnof	$\mathfrak{f}$	
710	circ	$\sim$	
732	tilde	$\sim$	
818	lowline	$\_$	(in LaTeX becomes a short underscore)
913	Alpha	$A$	
914	Beta	$B$	
915	Gamma	$\Gamma$	
916	Delta	$\Delta$	
917	Epsilon	$E$	
918	Zeta	$Z$	
919	Eta	$H$	
920	Theta	$\Theta$	
921	Iota	$I$	
922	Kappa	$K$	
923	Lambda	$\Lambda$	
924	Mu	$M$	

925	Nu	N	
926	Xi	$\Xi$	
927	Omicron	O	
928	Pi	$\Pi$	
929	Rho	P	
931	Sigma	$\Sigma$	
932	Tau	T	
933	Upsilon	$\Upsilon$	
934	Phi	$\Phi$	
935	Chi	X	
936	Psi	$\Psi$	
937	Omega	$\Omega$	
945	alpha	$\alpha$	
946	beta	$\beta$	
947	gamma	$\gamma$	
948	delta	$\delta$	
949	epsilon	$\epsilon$	
950	zeta	$\zeta$	
951	eta	$\eta$	
952	theta	$\theta$	
953	iota	$\iota$	
954	kappa	$\kappa$	
955	lambda	$\lambda$	
956	mu	$\mu$	
957	nu	$\nu$	
958	xi	$\xi$	
959	omicron	$o$	
960	pi	$\pi$	
961	rho	$\rho$	
962	sigmaf	$\varsigma$	
963	sigma	$\sigma$	
964	tau	$\tau$	
965	upsilon	$v$	
966	phi	$\varphi$	
967	chi	$\chi$	
968	psi	$\psi$	
969	omega	$\omega$	
977	thetasym	$\vartheta$	
978	upsih	$\Upsilon$	
981	phisym	$\phi$	
982	piv	$\varpi$	
8194	ensp		0.5em space in LaTeX
8195	emsp		1em space in LaTeX
8195	quad*		$\backslash$ quad in LaTeX
8196	thicksp*		$\backslash$ ; in LaTeX

8197	medsp*	$\backslash:$ in LaTeX
8201	thinsp	$\backslash,$ in LaTeX
8203	strut*	$\backslash\text{strut}$ in LaTeX
8203	pagebreak*	$\backslash\text{newpage}$ in LaTeX
8204	zwnj	
8205	zwj	
8206	lrm	
8207	rlm	
8211	ndash	—
8212	mdash	—
8216	lsquo	‘
8217	rsquo	’
8218	sbquo	,
8220	ldquo	“
8221	rdquo	”
8222	bdquo	„
8224	dagger	†
8225	Dagger	‡
8226	bull	•
8230	hellip	...
8240	permil	‰
8242	prime	/
8243	Prime	//
8249	lsaquo	◁
8250	rsaquo	▷
8254	oline	-
8260	frasl	.2
8364	euro	€
8450	CC	©
8469	NN	ℕ
8473	PP	ℙ
8474	QQ	ℚ
8477	RR	ℝ
8484	ZZ	ℤ
8465	image	ℑ
8472	weierp	℘
8476	real	℔
8482	trade	™
8501	alefsym	ℵ
8592	larr	←
8593	uarr	↑
8594	rarr	→
8595	darr	↓
8596	harr	↔
8629	crarr	↷

---

8656	lArr	$\Leftarrow$
8657	uArr	$\Uparrow$
8658	rArr	$\Rightarrow$
8659	dArr	$\Downarrow$
8660	hArr	$\Leftrightarrow$
8704	forall	$\forall$
8706	part	$\partial$
8707	exist	$\exists$
8709	empty	$\emptyset$
8711	nabla	$\nabla$
8712	isin	$\in$
8713	notin	$\notin$
8715	ni	$\ni$
8719	prod	$\prod$
8721	sum	$\sum$
8722	minus	$-$
8727	lowast	$*$
8730	radic	$\sqrt{\phantom{x}}$
8733	prop	$\propto$
8734	infin	$\infty$
8736	ang	$\angle$
8743	and	$\wedge$
8744	or	$\vee$
8745	cap	$\cap$
8746	cup	$\cup$
8747	int	$\int$
8756	there4	$\therefore$
8764	sim	$\sim$
8773	cong	$\cong$
8776	asyp	$\approx$
8800	ne	$\neq$
8801	equiv	$\equiv$
8804	le	$\leq$
8805	ge	$\geq$
8834	sub	$\subset$
8835	sup	$\supset$
8836	nsup	$\subsetneq$
8838	sube	$\subseteq$
8839	supe	$\supseteq$
8853	oplus	$\oplus$
8855	otimes	$\otimes$
8869	perp	$\perp$
8901	sdot	$\cdot$
8942	vellip	$\vdots$



---

8968	lceil	[
8969	rceil	]
8970	lfloor	[
8971	rfloor	]
9001	lang	<
9002	rang	>
9312	circled1	①
9313	circled2	②
9314	circled3	③
9315	circled4	④
9316	circled5	⑤
9317	circled6	⑥
9318	circled7	⑦
9319	circled8	⑧
9320	circled9	⑨
9321	circled10	⑩
9674	loz	◇
9824	spades	♠
9827	clubs	♣
9829	hearts	♥
9830	diams	◇
8617	hooklarr*	↵
8718	bbox*	■
9633	box*	□
9744	ballotbox*	□
9745	ballotc*	☑
9746	ballotx*	☒
10003	checkmark*	✓
10004	bcheckmark*	✔
10007	xmark*	✗
10008	bxmark*	✘
128270	mglass*	ℙ
NA	smallskip	small vertical space ( <code>\smallskip</code> in LaTeX)
NA	medskip	medium vertical space ( <code>\medskip</code> in LaTeX)
NA	bigskip	big vertical space ( <code>\bigskip</code> in LaTeX)

---

## A.10. Definitions of predefined custom blocks

All custom blocks are defined in the standard prelude. You can view this in the file selection dropdown in [madoko.net](https://madoko.net) or download the [online version](#)

## A.11. License and attribution

Madoko is free software and available under the [Apache 2.0 license](http://madoko.codeplex.com) from <http://madoko.codeplex.com>

Madoko uses various other libraries under various other licenses to extend its functionality.

- For Citation Style Language support, Madoko uses:
  - *Sax.js* (XML parser), Copyright (c) Isaac Z. Schlueter and Contributors, ISC License, <https://github.com/isaacs/sax-js> (String.fromCodePoint Copyright by Mathias Bynens, MIT License.)
  - *Bibtex-parser*, Copyright (c) 2010 Henrik Muehe and Mikola Lysenko and apcshields, MIT License, <https://github.com/apcshields/zotero-bibtex-parse>
  - *Citeproc.js*, Copyright (c) 2009-2014 Frank G. Bennett, Common Public Attribution License, <https://bitbucket.org/fbennett/citeproc-js>
  - *Locales-en-US.xml*, Creative Commons Attribution-ShareAlike 3.0, <https://github.com/citation-style-language/locales>
- The online environment Madoko.NET uses:
  - *Visual Studio.Code* (the editor component), Copyright (c) Microsoft Corporation, MIT License, <https://github.com/Microsoft/vscode>.
  - *Typo.js* (spell checker), Copyright (c) 2011 by Christopher Finke, Modified BSD License, <https://github.com/cfinke/Typo.js/>.
  - *The en-US dictionary*, 2006, This dictionary is based on a subset of the original English word list created by Kevin Atkinson for Pspell and Aspell and covered by his original LGPL license. The affix file is a heavily modified version of the original `english.aff` file which was released as part of Geoff Kuenning's Ispell and as such is covered by his BSD license.
  - *wcwidth.js* (unicode character width library), Copyright (C) 2012-2014 by Jun Woong and Tim Oxley. Based on the original C library by Markus Kuhn. MIT License, <https://github.com/mycoboco/wcwidth.js>.
- When rendering Madoko to PDF or rendering mathematics, various other programs are used that are usually installed with your TeX installation. Some important ones are:
  - *dvisvgm*, Copyright (c) Martin Giesekeing, <http://dvisvgm.bplaced.net/>
  - *dvipng*, Copyright (c) 2002-2015 Jan-Ake Larsson,

- `(xe,pdf)latex`, Copyright (c) Donald M. Knuth, Peter Breitenlohner (eTeX), Han The Thanh (pdfTeX), and Jonathan Kew and Khaled Hosny (XeTeX).

- This documentation uses the free Monarch butterfly image  from [Clkr-freeVectorImages](#).

Thanks to all the authors for their excellent libraries! Special thanks to Martin Giesekeing with his help to make `dvisvgm` work well with Madoko math formulas.