FLETCHER T. PENNEY

MULTIMARKDOWN USER'S GUIDE

Contents

MultiMarkdown User's Guide

Version 4.5.3
Revised 2014-06-23

Introduction

> As the world goes multi-platform with all of the new mobile operating systems, MultiMarkdown provides an easy way to share formatting between all of my devices. It's easy to learn (even for us mortals) and immediately useful.
>
> — David Sparks, MacSparky.com[1]

[1] http://MacSparky.com/

What is Markdown?

To understand what MultiMarkdown is, you first should be familiar with Markdown[2]. The best description of what Markdown is comes from John Gruber's Markdown web site:

[2] http://daringfireball.net/projects/markdown/

> Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).
>
> Thus, "Markdown" is two things: (1) a plain text formatting syntax; and (2) a software tool, written in Perl, that converts the plain text formatting to HTML. See the Syntax page for details pertaining to Markdown's formatting syntax. You can try it out, right now, using the online Dingus.
>
> The overriding design goal for Markdown's formatting syntax is to make it as readable as possible. The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. While Markdown's syntax has been influenced by several existing text-to-HTML filters, the single biggest source of inspiration for Markdown's syntax is the format of plain text email. — John Gruber[3]

[3] http://daringfireball.net/projects/markdown/

What is MultiMarkdown?

Markdown is great, but it lacked a few features that would allow it to work with documents, rather than just pieces of a web page.

I wrote MultiMarkdown in order to leverage Markdown's syntax, but to extend it to work with complete documents that could ultimately be converted from text into other formats, including complete

HyperText Markup Language (HTML) documents, LaTeX, PDF, ODF, or even (shudder) Microsoft Word documents.

In addition to the ability to work with complete documents and conversion to other formats, the Markdown syntax was lacking a few other things. Michel Fortin added a few additional syntax features when writing PHP Markdown Extra[4]. Some of his ideas were implemented and expanded on in MultiMarkdown, in addition to including features not available in other Markdown implementations. These features include tables, footnotes, citation support, image and link attributes, cross-references, math support, and more.

[4] http://www.michelf.com/projects/php-markdown/extra/

John Gruber may disagree with me, but I really did try to stick with his proclaimed vision whenever I added a new syntax format to MultiMarkdown. The quality that attracted me to Markdown the most was its clean format. Reading a plain text document written in Markdown is *easy*. It makes sense, and it looks like it was designed for people, not computers. To the extent possible, I tried to keep this same concept in mind when working on MultiMarkdown.

I may or may not have succeeded in this. . . .

In the vein of Markdown's multiple definitions, you can think of MultiMarkdown as:

1. A program to convert plain text to a fully formatted document.
2. The syntax used in the plain text to describe how to convert it to a complete document.

Why should I use MultiMarkdown?

Writing with MultiMarkdown allows you to separate the content and structure of your document from the formatting. You focus on the actual writing, without having to worry about making the styles of your chapter headers match, or ensuring the proper spacing between paragraphs. And with a little forethought, a single plain text document can easily be converted into multiple output formats without having to rewrite the entire thing or format it by hand. Even better, you don't have to write in "computer-ese" to create well formatted HTML or LaTeX commands. You just write, MultiMarkdown takes care of the rest.

For example, instead of writing:

```
<p>In order to create valid
<a href="http://en.wikipedia.org/wiki/HTML">HTML</a>, you
need properly coded syntax that can be cumbersome for
&#8220;non-programmers&#8221; to write. Sometimes, you
just want to easily make certain words <strong>bold
```

```
</strong>, and certain words <em>italicized</em> without
having to remember the syntax. Additionally, for example,
creating lists:</p>

<ul>
<li>should be easy</li>
<li>should not involve programming</li>
</ul>
```

You simply write:

```
In order to create valid [HTML], you need properly
coded syntax that can be cumbersome for
"non-programmers" to write. Sometimes, you just want
to easily make certain words **bold**, and certain
words *italicized* without having to remember the
syntax. Additionally, for example, creating lists:

* should be easy
* should not involve programming

[HTML]: http://en.wikipedia.org/wiki/HTML
```

Additionally, you can write a MultiMarkdown document in any text editor, on any operating system, and know that it will be compatible with MultiMarkdown on any other operating system and processed into the same output. As a plain text format, your documents will be safe no matter how many times you switch computers, operating systems, or favorite applications. You will always be able to open and edit your documents, even when the version of the software you originally wrote them in is long gone.

These features have prompted several people to use MultiMarkdown in the process of writing their books, theses, and countless other documents.

There are many other reasons to use MultiMarkdown, but I won't get into all of them here.

By the way — the MultiMarkdown web site is, of course, created using MultiMarkdown. To view the MMD source for any page, add `.txt` to the end of the URL. If the URL ends with `/`, then add `index.txt` to the end instead. The main MultiMarkdown page, for example, would be http://fletcherpenney.net/multimarkdown/index.txt.

What Are the Different Versions of MultiMarkdown?

The first real version of MultiMarkdown was version 2. It was a modification of the original `Markdown.pl` script. It worked fine, but was slow when parsing longer documents. The plain text was converted to HTML, and then XSLT was used to convert the HTML to other formats (primarily LaTeX). Over time, maintaining the complicated nest of regular expressions became more difficult, and a better approach was needed.

MultiMarkdown 3 (aka `peg-multimarkdown`) was built using John MacFarlane's peg-markdown[5] as a base. It was *much* faster than version 2, and the underlying PEG (parsing expression grammar) made things more reliable. There were still issues and limitations (some inherited from peg-markdown, but most were my errors), which lead to the development of version 4.

[5] https://github.com/jgm/peg-markdown

MultiMarkdown 4[6] was a complete rewrite, keeping only the PEG and a few utility routines from MMD v3. This release fixed memory leaks and other problems from earlier MMD releases; it is safe to use in multithreaded applications and adds many new features. By far, it's the best version to date!

[6] http://github.com/fletcher/MultiMarkdown-4

Where is this Guide Kept?

This guide has been rewritten with the following changes:

- The source is now in the `gh_pages` branch of the MultiMarkdown project[7]. You can submit changes as a pull request, or by writing me.

[7] https://github.com/fletcher/MultiMarkdown-4

- You can access this information on the web at http://fletcher.github.io/MultiMarkdown-4
- The source itself is a collection of MultiMarkdown text documents that use the transclusion features to create a master document from the individual source files. These documents can be viewed in the browser as HTML, or downloaded as PDF or OpenDocument files.

The Philosophy Behind MultiMarkdown

My vision for MultiMarkdown was inspired by my understanding of what made Markdown so wonderful. Markdown is simple. It's easy to remember. It's intuitive to read. Markdown avoids the "everything but the kitchen sink" problem.

My goal for MultiMarkdown is that it should be useable for 80% of the documents that 80% of people write. Obviously that is not a precise estimate, but the idea is that *most* people can write *most* of their documents using it. Some people can write everything in MMD. Some people can write very little in MMD. MultiMarkdown (by itself) would not be very good for writing a comic book, for example. It's perfect for writing a novel.

A central tenet of MultiMarkdown is that the focus is on *content*, not *presentation*. I honestly couldn't care whether you want to use Arial, Helvetica, or Comic Sans for your masterpiece. The presentation/styling/appearance is for you to decide. You pick the fonts. You pick the colors. What I care about with MultiMarkdown is that most (not necessarily all) of the *meaning* of the document is represented — this is a list, that is a table, this is a top-level heading, etc.

A well written MultiMarkdown document will look reasonably good whether you output to HTML, LaTeX, OpenDocument, etc. It might not look perfect. A page might break at an inopportune place. The title page of a LaTeX document doesn't have an exact analogy in HTML. HTML doesn't handle page breaks well.

If you're writing your thesis, publishing a book, or submitting a document to the board of directors — by all means write in MultiMarkdown. Focus on the content and overall structure. And when you're ready, convert to your desired output format. Proofread. And when you're sure that you like what you've got, *then* focus on the aesthetics. Insert a page break. Tweak fonts. Go wild. But do it in a tool appropriate for the format you're using. This might be a good programmer's text editor for HTML and CSS. It might be LyX[8]. It might be LibreOffice[9].

[8] http://www.lyx.org/

[9] http://www.libreoffice.org/

The Purpose of MultiMarkdown

In the years since MultiMarkdown was first released, I've received countless emails of all kinds. A group stands out that seems to point to a philosophical difference between types of users.

Computers are wonderful for doing the tedious sorts of things that humans tend to not enjoy and to suck at. For example, I don't want to have to add all the columns in a spreadsheet by hand. That's what computers are for. Conversely, I don't want to read a novel written by a computer (at least not yet. . .)

As applied to MultiMarkdown, it's purpose is to handle the tedium of applying repetitive formatting rules to text. For example, having to wrap every single paragraph in <p> tags for a web site is really tedious. The computer should be able to handle that easily.

But the user should still understand *why* those <p> tags are necessary. The goal of MultiMarkdown is not to say, "Don't worry your pretty little head about complicated things like HTML or LaTeX." The goal is to allow you to learn and appreciate things like HTML and LaTeX without most of the tedium that goes along with marking up a document by hand.

MultiMarkdown is not a magical "black box" that converts plain text to HTML with a lot of hand-waving and "pay no attention to the man behind the curtain." Programs that use this approach tend to result in crappy output (e.g. Microsoft products, most apps to create web pages "for you", etc.)

So when you're trying to do something fancy, or trying to troubleshoot a problem, start at the end. Look at the HTML/LaTeX/whatever that is generated and see what's going on at a fundamental level. Once you understand that, then look at what MultiMarkdown is doing. I believe you'll have an easier time solving problems, and probably learn a thing or two along the way. . . .

Feature Requests

I often get feature requests. Some requests are really good ideas and I implement them. Some are really good ideas and I don't implement them. Some, however, miss the point of MultiMarkdown entirely.

I completely understand that somewhere out there, somebody's life would be complete if MultiMarkdown had a feature that drew a picture of a bunny after every 15th word of a MultiMarkdown document. But that feature would be absolutely useless to everyone else on the planet. Add enough of those sorts of features, and you end up with Microsoft Word. Which I am sure is the only application that some people are able to use, *precisely because* it draws bunnies,

and even lets you choose which color and breed of bunny to use. And whether the bunny is left- or right-pawed.

I am not going to program MultiMarkdown to draw bunnies.

Instead, if there is something that you wish MultiMarkdown would do, consider the following:

1. First, make sure the feature you want doesn't already exist. Read the documentation. Look at the Sample Gallery[10].

2. If it's not there, consider whether you can "hijack" an existing feature. For example, I needed to create a PDF to print a book of poetry. MMD didn't have a "poetry" feature. But it did have code blocks, which are essentially the same thing, except poetry doesn't usually use monospaced fonts. Voila, I used code blocks for all of the poems, and then changed the LaTeX output to refrain from using monospaced fonts when displaying code.

3. Still stuck? — ask for help. The discussion list[11] is a great place to get help, as is the support site[12].

4. But before requesting a new feature, honestly ask yourself how many other people need it. Ask yourself why, if it's such a great idea, it hasn't been implemented yet.

5. If your idea isn't really useful to other people, then that makes it the *perfect* opportunity to develop some new skills. Grab a copy of the MultiMarkdown source, and start hacking away to add what you want. Test it out. If it's really great, share it on the discussion list to let others use it. Convince everyone that it simply *must* be added to the core source.

[10] https://github.com/fletcher/MultiMarkdown-Gallery

[11] https://groups.google.com/forum/#!forum/multimarkdown

[12] http://support.fletcherpenney.net/

Installation

You have several options for obtaining and installing MultiMarkdown:

- Binary installer – available for:
  - Mac OS
  - Windows
- Use a third party package installer:
  - `homebrew` for Mac OS
- Compile from source – useful if you want to modify MMD's behavior, or if there isn't an available binary download for your operating system

Mac OS

You can download the installers from the MMD website download[13] page. You need the `Mac Installer`. Download it. Run it. Done.

If you use older tools that were designed for MultiMarkdown version 3, you may need to use the `Mac Support Installer`. This is also useful if you need the older XSLT based parsing tools.

If you plan on creating LaTeX documents, you should also download the LaTeX Support Files[14] and install them into the appropriate location for your system and LaTeX software.

[13] http://fletcherpenney.net/multimarkdown/download/

[14] https://github.com/fletcher/peg-multimarkdown-latex-support

**nix*

Unix/Linux users should use the instructions for compiling from source below.

Windows

The easiest way to install MMD on Windows is the `MultiMarkdown-Windows` installer from the download[15] page and run it. The installer is built

[15] http://fletcherpenney.net/multimarkdown/download/

using software by BitRock[16].

Just as with the Mac OS X version, the installer includes the `multimarkdown` binary, as well as several convenience scripts.

You can use Windows Explorer to create shortcuts to the `multimarkdown` binary, and adjust the properties to allow you to create "drag and drop" versions of MMD as well.

You can also download a "Portable" version that can be run off USB thumb drives, for example. It is also available on the download[17] page.

[16] http://bitrock.com/

[17] http://fletcherpenney.net/multimarkdown/download/

Free BSD

If you want to compile manually, you should be able to follow the directions for Linux below. However, apparently MultiMarkdown has been put in the ports tree, so you can also use:

```
cd /usr/ports/textproc/multimarkdown
make install
```

(I have not tested this myself, and cannot guarantee that it works properly.)

Compile From Source

For Mac and *nix users:

- Download the source from the github[18] web site using `git https://github.com/fletcher/MultiMarkdown-4.git`
- `git submodule init` and then `git submodule update` to download `greg` and the test suite submodules (as well as others)
- Run `make` to compile.
- Run `make test-all | less` (or `make test-all | grep failed` for a more concise version) to verify that the build is correct. One of the tests is expected to fail ("Ordered and unordered lists"); the rest should pass on all systems.
- `make install` (as root) will install the software
- `make install-scripts` will install the helper scripts for you (e.g.`mmd`, `mmd2tex`, etc.)
- If you plan on creating LaTeX documents, you should also download the LaTeX Support Files[19] and install them into the appropriate location for your system and LaTeX software.

[18] https://github.com/fletcher/MultiMarkdown-4

[19] https://github.com/fletcher/peg-multimarkdown-latex-support

MultiMarkdown includes a few other projects as submodules, but the only one you need to actually compile the code is the `greg` software. Once compiled, MultiMarkdown has no external dependencies – the binary is self-contained. Therefore, it should basically compile and run anywhere.

Windows users can obtain the code in the same way, but will need to use their own compiler. The way I compile for Windows is actually to use the `make windows` command running on a *nix system with MinGW installed.

How to Use MultiMarkdown

There are several ways to use MultiMarkdown, depending on your needs. You can use the `multimarkdown` command line tool, you can use MultiMarkdown with several applications that support it directly, or you can use a drag and drop approach.

Command Line Usage

First, verify that you have properly installed MultiMarkdown:

```
multimarkdown -v
```

If you don't see a message telling you which version of MultiMarkdown is installed, check out Troubleshooting.

To learn more about the command line options to MultiMarkdown:

```
multimarkdown -h
```

Once you have properly installed MultiMarkdown:

```
multimarkdown file.txt
```

will convert the plain text file `file.txt` into HTML output. To save the results to a file:

```
multimarkdown file.txt > file.html
```

A shortcut to this is to use MultiMarkdown's batch mode, which will save the output to the same base filename that is input, with the extension `.html` (or `.tex` for LaTeX output):

```
multimarkdown -b file.txt
```

A benefit of batch mode is that you can process multiple files at once:

```
multimarkdown -b file1.txt file2.txt file3.txt
```

If you want to create LaTeX output instead of HTML:

```
multimarkdown -t latex file.txt
```

For LyX:

```
multimarkdown -t lyx file.txt
```

For OPML:

```
multimarkdown -t opml file.txt
```

For RTF (RTF output is limited – check the output carefully to be sure it's ok for your needs):

```
multimarkdown -t rtf file.txt
```

And for an OpenDocument text file:

```
multimarkdown -t odf file.txt
```

There are also several convenience scripts included with MultiMarkdown:

```
mmd file.txt
mmd2tex file.txt
mmd2opml file.txt
mmd2odf file.txt
```

These scripts run MultiMarkdown in batch mode to generate HTML, LaTeX, OPML, or ODF files respectively. These scripts are included with the Mac or Windows installers, and are available for *nix in the `scripts` directory in the source project. They are intended to be used as shortcuts for the most common command line options.

Command Line Options

There are several options when running MultiMarkdown from the command line.

```
multimarkdown -h, multimarkdown --help
```

This shows a summary of how to use MultiMarkdown.

```
multimarkdown -v, multimarkdown --version
```

Displays the version of MultiMarkdown currently installed.

```
multimarkdown -o, multimarkdown --output=FILE
```

Directs the output to the specified file. By default, the output is directed to `stdout`. The use of `batch` mode obviates the need to use this option, but if you want to specify a different output filename it can be handy.

```
multimarkdown -t html|latex|memoir|beamer|opml|odf|rtf|lyx|lyx-beamer
```

This options specified the format that MultiMarkdown outputs. The default is html. If you use the `LaTeX Mode` metadata, then MultiMarkdown will automatically choose `memoir` or `beamer` as directed without using these command line options. Using that metadata will also allow the various convenience scripts to choose the correct output format as well.

```
multimarkdown -b, multimarkdown --batch
```

Automatically redirects the output to a file with the same base name as the input file, but with the appropriate extension based on the output type. For example, `multimarkdown -b file.txt` would output the HTML to `file.html`, and `multimarkdown -b -t latex file.txt` would output to `file.tex`.

```
multimarkdown -c, multimarkdown --compatibility
```

Compatibility mode causes MultiMarkdown to output HTML that is compatible with that output from the original Markdown. This allows it to pass the original Markdown test suite. Syntax features

that don't exist in regular Markdown will still be output using the regular MultiMarkdown output formatting.

```
multimarkdown -f, multimarkdown --full
```

The `full` option forces a complete document, even if it does not contain enough metadata to otherwise trigger a complete document.

```
multimarkdown -s, multimarkdown --snippet
```

The `snippet` option forces the output of a "snippet", meaning that header and footer information is left out. This means that a LaTeX document might not have enough information to be processed, for example.

```
multimarkdown --process-html
```

This option tells MultiMarkdown to process the text included within HTML tags in the source document. This can feature can also be implemented on a tag-by-tag basis within the document itself, such as `<div markdown="1">`.

```
multimarkdown -m, multimarkdown --metadata-keys
```

List all of the available metadata keys contained in a document, one key per line.

```
multimarkdown -e "metakey", multimarkdown --extract "metakey"
```

The extract feature outputs the value of the specified metadata key. This is used in my convenience scripts to help choose the proper LaTeX output mode, and could be used in other circumstances as well.

```
multimarkdown --random
```

Tell MultiMarkdown to use random identifier numbers for footnotes. Useful when you might combine multiple HTML documents together, e.g. in a weblog.

```
multimarkdown --accept
multimarkdown --reject
```

```
multimarkdown --accept --reject
```

Tell MultiMarkdown whether to accept or reject changes in written in CriticMarkup[20] format within the document. Use both together if you want to highlight the differences – this only works for HTML output.

[20] http://criticmarkup.com/

```
multimarkdown --smart
multimarkdown --nosmart
```

Tell MultiMarkdown whether to use "smart" typography, similar to John Gruber's SmartyPants[21] program, which was included in MultiMarkdown 2.0. This extension is turned on by default in MultiMarkdown.

[21] http://daringfireball.net/projects/smartypants/

```
multimarkdown --notes
multimarkdown --nonotes
```

Tell MultiMarkdown whether to use footnotes (enabled by default).

```
multimarkdown --labels
multimarkdown --nolabels
```

Tell MultiMarkdown whether to add id attributes to headers in HTML (enabled by default).

```
multimarkdown --mask
multimarkdown --nomask
```

Tell MultiMarkdown whether to mask email addresses when creating HTML (enabled by default).

```
multimarkdown --notes
```

Enables the use of footnotes and similar markup (glossary, citations). Enabled by default in MultiMarkdown.

Other options are available by checking out `multimarkdown --help-all`, but the ones listed above are the primary options.

Advanced Mode

MultiMarkdown version 2.0 had to first convert the source file to HTML, and then applied XSLT files to convert to the final LaTeX format. Since MultiMarkdown 3.0 can create LaTeX directly, this approach is no longer necessary.

The one benefit of that approach, however, was that it became possible to perform a wide range of customizations on exactly how the LaTeX output was created by customizing the XSLT files.

If you install the Support files on Mac or Linux, you can still use the advanced XSLT method to generate LaTeX output. For the time being, this approach doesn't work with Windows, but it would be fairly easy to create a batch script or perl script to implement this feature on Windows.

Keep in mind, however, that because of the more advanced mechanism of handling LaTeX in MultiMarkdown 3.0, you can do a great deal of customization without needing to use an XSLT file.

The `mmd2tex-xslt` script will convert a plain text file into LaTeX that is virtually identical with that created by the regular LaTeX approach.

There are a few differences in the two approaches, however:

- Once a MultiMarkdown file is converted to HTML, it is impossible to tell whether the resulting HTML was generated by MultiMarkdown, or if it was included as raw HTML within the source document. So *either* way, it will be converted to the analagous LaTeX syntax. The `multimarkdown` binary on its own will *not* convert HTML into LaTeX.

- The whitespace that is generated will be different under certain circumstances. Typically, this will result in one extra or one fewer blank lines with the the XSLT approach. Generally this will not be an issue, but when used with `<!-- some comment -->` it may cause a newline to be lost.

- The default XSLT recognizes `class="noxslt"` when applied to HTML entities, and will discard them from the output.

- An XSLT can only be applied to a complete HTML document, not a "snippet". Therefore, if you want to use the XSLT method, your file must have metadata that triggers a complete document (i.e. any metadata except "quotes language" or "base header level").

- Using XSL to process an HTML file will "de-obfuscate" any email addresses that were obfuscated by MultiMarkdown.

Recommendations

I recommend that you become familiar with the "basic" approach to using MultiMarkdown before trying to experiment with XSLT. The basic approach is faster, and easier, and the results can still be customized quite a bit.

Then you can experiment with modifying XSLT to further customize your output as needed.

If you have XSLT files that you used in MultiMarkdown 2.0, you will likely need to modify them to recognize the HTML output generated by MultiMarkdown 3.0. You can use the default XSLT files as a guide to what is different.

Mac OS X Applications

There are several applications that have built-in support for MultiMarkdown, or that can easily use it with a plug-in.

Using MultiMarkdown With MultiMarkdown Composer

MultiMarkdown Composer[22] is my commercial text editor designed from the ground up around the MultiMarkdown (and Markdown) syntax. It contains a great deal of features to make writing, editing, and exporting MultiMarkdown documents easier than ever before. I certainly recommend it, but since I created it, and it's not free, you may believe me to biased. So search the internet to see what people are saying, then check it out.

[22] http://multimarkdown.com/

Using MultiMarkdown with TextMate

If you want to run MultiMarkdown from directly within TextMate[23], you should install my MultiMarkdown bundle[24]. This is a modified version of the original Markdown bundle for TextMate that includes better support for MultiMarkdown.

This bundle will work with MultiMarkdown 2, or with MultiMarkdown 3/4 if you install the Mac Support Installer files (available from the downloads page[25]).

[23] http://macromates.com/

[24] https://github.com/fletcher/markdown.tmbundle

[25] http://fletcherpenney.net/multimarkdown/download/

Using MultiMarkdown with Scrivener

Scrivener[26] is a great program for writers using Mac OS X. It includes built in support for MultiMarkdown. If you want to use MultiMarkdown 3/4 with Scrivener, you need to install the Support files in `~/Library/Application Support/MultiMarkdown`. The Mac Support Installer is available from the downloads page[27] and will install these

[26] http://www.literatureandlatte.com/

[27] http://fletcherpenney.net/multimarkdown/download/

files for you.

Drag and Drop

You can use the Mac OS X drag and drop applications to allow you to convert MultiMarkdown to other formats by dragging and dropping files in the Finder. They are available from the download[28] page, or by running `make drop` from the command line in the `multimarkdown` source directory.

[28] http://fletcherpenney.net/multimarkdown/download/

MultiMarkdown and Finder "Quick Look"

Starting in Mac OS 10.5, the Finder has the ability to show a "Quick Look" preview of the contents of a file. I have a Quick Look generator that allows the Finder to preview the contents of a MultiMarkdown text file (or OPML file) as an HTML preview.

I recommend using the latest (closed-source) version available for download[29]. It contains advanced features that are not available in the open source version.

[29] http://multimarkdown.com/download/

Source code for the older version is available for download from github[30].

[30] https://github.com/fletcher/MMD-QuickLook

Using MultiMarkdown in Windows

You can use the same command line approach with Windows as described previously. While there aren't drag and drop applications per se for the Windows system, you can use Windows Explorer to create links to the binary and specify and desired command line options to change the default output format. This will effectively allow you to create drag and drop applications for Windows.

MultiMarkdown and LaTeX

Of note LaTeX[31] is a complex set of programs. MultiMarkdown doesn't include LaTeX in the installer — it's up to the user to install a working LaTeX setup on their machine if you want to use it.

[31] http://en.wikipedia.org/wiki/LaTeX

What MultiMarkdown does is make it easier to generate documents using the LaTeX syntax. It should handle 80% of the documents that 80% of MultiMarkdown need. It doesn't handle all circumstances, and sometimes you will need to hand code your LaTeX yourself.

In those cases you have a few options. MultiMarkdown will pass text included in HTML comments along to the LaTeX as raw output. For example:

```
<!--  This is raw \LaTeX \[ {e}^{i\pi }+1=0 \] -->
```

You can also include your desired LaTeX code in a separate file and link to it:

```
<!-- \input{somefile} -->
```

If you have questions about LaTeX itself, I can't help. You're welcome to send your question to the MultiMarkdown discussion list[32], and perhaps someone will be able to offer some assistance. But you would be better off asking a group dedicated to LaTeX instead.

If the problem is that MultiMarkdown itself is generating invalid LaTeX, then of course I want to know about it so I can fix it.

If you need more information about how to use LaTeX to process a file into a PDF, check out the faq (**??**).

[32] https://groups.google.com/forum/#!forum/multimarkdown

MultiMarkdown and OPML

MultiMarkdown is well suited to plain text files, but it can also be useful to work on MultiMarkdown documents in an outliner or mind-mapping application. For this, it is easy to convert back and forth between OPML and plain text MultiMarkdown.

To convert from a text file to OPML:

```
multimarkdown -t opml -b file.txt
```

or:

```
mmd2opml file.txt
```

The resulting OPML file uses the headings to build the outline structure, and puts the text within each section as a not for the corresponding level of the outline using the `_note` attribute. **NOTE**: not all outliners support this attribute. On Mac OS X, OmniOutliner[33] is a fabulous outliner that supports this field. If you're into mind mapping software, iThoughts[34] works on the iPad/iPhone and supports import and export with OPML and the `_note` attribute.

To convert from OPML, you can use various commands in from the MMD-Support[35] package:

```
opml2HTML file.opml
opml2mmd file.opml
```

[33] http://www.omnigroup.com/applications/omnioutliner/

[34] http://www.ithoughts.co.uk/

[35] https://github.com/fletcher/MMD-Support

```
opml2LaTeX file.opml
```

NOTE: These scripts require a working installation of `xsltproc`, and the ability to run shell scripts. This should work by default on most installations of Mac OS X or Linux, but will require these applications to be installed separately on Windows.

MultiMarkdown and OpenDocument

It is also possible to convert a MultiMarkdown text file into a word processing document for OpenOffice.org[36] or LibreOffice[37]. This file can then be converted by one of those applications into RTF, or a Microsoft Word document, or many other file formats. (If you're not familiar with these applications, they are worth checking out. I don't understand why people use Microsoft Office any more. . .)

[36] http://www.openoffice.org/

[37] http://www.libreoffice.org/download

```
multimarkdown -b -t odf file.txt
```

or

```
mmd2odf file.txt
```

MultiMarkdown 2.0 had partial support for outputting an RTF file, and could do it completely on Mac OS X by using Apple's `textutil` program. MMD 3 no longer directly supports RTF as an output format, but the Flat OpenDocument format is a much better option.

NOTE: LibreOffice can open these Flat OpenDocument files by default, but OpenOffice requires that you install the `OpenDocument-Text-Flat-XML.jar` file available from the downloads[38] page. To install it, create a new document in OpenOffice (or open an existing one), then go to the Tools->XML Filter Settings menu option. Use the "Open Package. . . " button to import the downloaded `.jar` file.

[38] https://github.com/fletcher/peg-multimarkdown/downloads

MultiMarkdown and RTF

I have made it clear in various places that RTF is a horrible format for sharing documents. Seriously – it's really bad.

That said, MultiMarkdown now offers direct conversion to RTF documents (sort of). This export format is not complete. Tables don't work very well, and lists don't work properly. Images are not supported.

If you have a very simple document, this may work just fine.

If you have a more complex document, I encourage you to use

the OpenDocument export, and to use LibreOffice[39] instead of a commercial Word-processor (you know what I'm talking about). Even if you use LibreOffice to convert your OpenDocument to RTF, you'll get better results.

[39] http://www.libreoffice.org/

MultiMarkdown and LyX

LyX[40] is is a document processor that seems to be a sort of hybrid between a markup language processor and a word processor. I'll be honest – I don't quite get it, and I don't use it.

That said, Charles Cowan has contributed code to the MultiMarkdown project that enables exporting of LyX documents directly. If you have any trouble getting this to work, please use the MultiMarkdown issues page[41] to get help.

See his page[42] for more information.

Note: Because the LyX exporter is not maintained by me, it may take some time for new features to be supported when exporting to LyX.

[40] http://www.lyx.org/

[41] https://github.com/fletcher/MultiMarkdown-4/issues

[42] http://crcowan.github.io/MultiMarkdown-4-LyX-Maintenance/

Advanced Use

It is possible to use an XSLT file to customize the OpenDocument output from MultiMarkdown. I suppose you could also write an XSLT to convert OpenDocument into LaTeX, similar to the default ones that convert HTML into LaTeX.

You can also create an XSLT that converts the OpenDocument output and modifies it to incorporate necessary customizations. While a little tricky to learn, XSLT files can be quite powerful and you're limited only by your imagination.

Limitations

There are several limitations to the OpenDocument Flat Text format:

- images are not fully supported — they work best if you specify a length and a width in "fixed" units (not '%'), or do not specify any dimensions.
- citations are not supported — I would like to be able to do something here, but I suspect you will need to use an external tool for the time being
- math features are not supported, though I hope to be able to implement this at some point in the future

Syntax

Metadata

It is possible to include special metadata at the top of a MultiMarkdown document, such as title, author, etc. This information can then be used to control how MultiMarkdown processes the document, or can be used in certain output formats in special ways. For example:

```
Title:    A Sample MultiMarkdown Document
Author:   Fletcher T. Penney
Date:     February 9, 2011
Comment:  This is a comment intended to demonstrate
          metadata that spans multiple lines, yet
          is treated as a single value.
CSS:      http://example.com/standard.css
```

The syntax for including metadata is simple.

- The metadata must begin at the very top of the document - no blank lines can precede it. There can optionally be a `---` on the line before and after the metadata. The line after the metadata can also be `...`. This is to provide better compatibility with YAML[43], though MultiMarkdown doesn't support all YAML metadata.
- Metadata consists of the two parts - the `key` and the `value`
- The metadata key must begin at the beginning of the line. It must start with an ASCII letter or a number, then the following characters can consist of ASCII letters, numbers, spaces, hyphens, or underscore characters.
- The end of the metadata key is specified with a colon (':')
- After the colon comes the metadata value, which can consist of pretty much any characters (including new lines). To keep multiline metadata values from being confused with additional metadata, I recommend indenting each new line of metadata. If your

[43] http://www.yaml.org/

metadata value includes a colon, it *must* be indented to keep it from being treated as a new key-value pair.

- While not required, I recommend using two spaces at the end of each line of metadata. This will improve the appearance of the metadata section if your document is processed by Markdown instead of MultiMarkdown.

- Metadata keys are case insensitive and stripped of all spaces during processing. This means that `Base Header Level`, `base headerlevel`, and `baseheaderlevel` are all the same.

- Metadata is processed as plain text, so it should *not* include MultiMarkdown markup. It is possible to create customized XSLT files that apply certain processing to the metadata value, but this is not the default behavior.

- After the metadata is finished, a blank line triggers the beginning of the rest of the document.

Metadata "Variables"

You can substitute the `value` for a metadata `key` in the body of a document using the following format, where `foo` and `bar` are the `keys` of the desired metadata.

```
# A Variable in a Heading [%foo] #

A variable in the body [%bar].
```

"Standard" Metadata keys

There are a few metadata keys that are standardized in MultiMarkdown. You can use any other keys that you desire, but you have to make use of them yourself.

My goal is to keep the list of "standard" metadata keys as short as possible.

Author

This value represents the author of the document and is used in LaTeX, ODF, and RTF documents to generate the title information.

Affiliation

This is used to enter further information about the author — a link to a website, the name of an employer, academic affiliation, etc.

Base Header Level

This is used to change the top level of organization of the document. For example:

```
Base Header Level: 2

# Introduction #
```

Normally, the Introduction would be output as <h1> in HTML, or `\part{}` in LaTeX. If you're writing a shorter document, you may wish for the largest division in the document to be <h2> or `\chapter{}`. The `Base Header Level` metadata tells MultiMarkdown to change the largest division level to the specified value.

This can also be useful when combining multiple documents. `Base Header Level` does not trigger a complete document.

Additionally, there are "flavors" of this metadata key for various output formats so that you can specify a different header level for different output formats — e.g. `LaTeX Header Level`, `HTML Header Level`, and `ODF Header Level`.

If you are doing something interesting with File Transclusion (section), you can also use a negative number here. Since metadata is not used when a file is "transcluded", this allows you to use a different level of headings when a file is processed on its own.

Biblio Style

This metadata specifies the name of the BibTeX style to be used, if you are not using natbib.

BibTeX

This metadata specifies the name of the BibTeX file used to store citation information. Do not include the trailing '.bib'.

Copyright

This can be used to provide a copyright string.

CSS

This metadata specifies a URL to be used as a CSS file for the produced document. Obviously, this is only useful when outputting to HTML.

Date

Specify a date to be associated with the document.

HTML Header

You can include raw HTML information to be included in the header. MultiMarkdown doesn't perform any validation on this data — it just copies it as is.

As an example, this can be useful to link your document to a working MathJax installation (not provided by me):

```
HTML header:  <script type="text/javascript"
    src="http://example.net/mathjax/MathJax.js">
    </script>
```

Quotes Language

This is used to specify which style of "smart" quotes to use in the output document. The available options are:

- dutch (or `nl`)
- english
- french (`fr`)
- german (`de`)
- germanguillemets
- swedish (`sv`)

The default is `english` if not specified. This affects HTML output. To change the language of a document in LaTeX is up to the individual.

`Quotes Language` does not trigger a complete document.

LaTeX Author

Since MultiMarkdown syntax is not processed inside of metadata, you can use the `latex author` metadata to override the regular author metadata when exporting to LaTeX.

This metadata *must* come after the regular `author` metadata if it is also being used.

LaTeX Footer

A special case of the `LaTeX Input` metadata below. This file will be linked to at the very end of the document.

LaTeX Input

When outputting a LaTeX document it is necessary to include various directions that specify how the document should be formatted. These are not included in the MultiMarkdown document itself — instead they should be stored separately and linked to with `\input{file}` commands.

These links can be included in the metadata section. The metadata is processed in order, so I generally break my directives into a group that need to go before my metadata, a group that goes after the metadata but before the document itself, and a separate group that goes at the end of the document, for example:

```
latex input:        mmd-memoir-header
Title:              MultiMarkdown Example
Base Header Level:  2
latex mode:         memoir
latex input:        mmd-memoir-begin-doc
latex footer:       mmd-memoir-footer
```

You can download the LaTeX Support Files[44] if you want to output documents using the default MultiMarkdown styles. You can then use these as examples to create your own customized LaTeX output.

[44] https://github.com/fletcher/peg-multimarkdown-latex-support

This function should allow you to do almost anything you could do using the XSLT features from MultiMarkdown 2.0. More importantly, it means that advanced LaTeX users do not have to learn XSLT to customize their code as desired.

LaTeX Mode

When outputting a document to LaTeX, there are two special options that change the output slightly — `memoir` and `beamer`. These options

are designed to be compatible with the LaTeX classes of the same names.

LaTeX Title

Since MultiMarkdown syntax is not processed inside of metadata, you can use the `latex title` metadata to override the regular title metadata when exporting to LaTeX.

This metadata *must* come after the regular `title` metadata if it is also being used.

MMD Footer

The `MMD Footer` metadata is used to specify the name of a file that should be appended to the end of the document using the File Transclusion (section) feature. This is useful for keeping a list of references, abbreviations, footnotes, links, etc. all in a single file that can be reused across multiple documents. If you're building a big document out of smaller documents, this allows you to use one list in all files, without multiple copies being inserted in the master file.

ODF Header

You can include raw XML to be included in the header of a file output in OpenDocument format. It's up to you to properly format your XML and get it working — MultiMarkdown just copies it verbatim to the output.

Title

Self-explanatory.

Transclude Base

When using the File Transclusion (section) feature to "link" to other documents inside a MultiMarkdown document, this metadata specifies a folder that contains the files being linked to. If omitted, the default is the folder containing the file in question. This can be a relative path or a complete path.

This metadata can be particularly useful when using MultiMarkdown to parse a text string that does not exist as a file on the computer, and therefore does not have a parent folder (when using `stdin` or another application that offers MultiMarkdown support). In this case, the path must be a complete path.

Smart Typography

MultiMarkdown incorporates John Gruber's SmartyPants[45] tool in addition to the core Markdown functionality. This program converts "plain" punctuation into "smarter" typographic punctuation.

[45] http://daringfireball.net/projects/smartypants/

Just like the original, MultiMarkdown converts:

- Straight quotes (`"` and `'`) into "curly" quotes
- Backticks-style quotes (`"this"`) into "curly" quotes
- Dashes (`--` and `---`) into en- and em- dashes
- Three dots (`...`) become an ellipsis

MultiMarkdown also includes support for quotes styles other than English (the default). Use the `quotes language` metadata to choose:

- dutch (`nl`)
- german(`de`)
- germanguillemets
- french(`fr`)
- swedish(`sv`)

This feature is enabled by default, but is disabled in `compatibility` mode, since it is not part of the original Markdown. You can also use the `smart` and `nosmart` command line options to control this feature.

Cross-References

An oft-requested feature was the ability to have Markdown automatically handle within-document links as easily as it handled external links. To this aim, I added the ability to interpret `[Some Text][]` as a cross-link, if a header named "Some Text" exists.

As an example, `[Metadata][]` will take you to the section describing metadata (section).

Alternatively, you can include an optional label of your choosing to help disambiguate cases where multiple headers have the same title:

```
### Overview [MultiMarkdownOverview] ##
```

This allows you to use `[MultiMarkdownOverview]` to refer to this section specifically, and not another section named `Overview`. This works with atx- or settext-style headers.

If you have already defined an anchor using the same id that is used by a header, then the defined anchor takes precedence.

In addition to headers within the document, you can provide labels for images and tables which can then be used for cross-references as well.

Link and Image Attributes

Adding attributes to links and images has been requested for a long time on the Markdown discussion list. I was fairly opposed to this, as most of the proposals really disrupted the readability of the syntax. I consider myself a "Markdown purist", meaning that I took John's introduction to heart:

> The overriding design goal for Markdown's formatting syntax is to make it as readable as possible. The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. While Markdown's syntax has been influenced by several existing text-to-HTML filters, the single biggest source of inspiration for Markdown's syntax is the format of plain text email.

Because there was not a syntax proposal that I felt fit this goal, I was generally opposed to the idea.

Then, Choan C. Gálvez proposed[46] a brilliantly simple syntax that stayed out of the way. By simply appending the attributes to the link reference information, which is already removed from the text itself, it doesn't disturb the readability.

[46] http://six.pairlist.net/pipermail/markdown-discuss/2005-October/001578.html

For example:

```
This is a formatted ![image][] and a [link][] with attributes.

[image]: http://path.to/image "Image title" width=40px height=400px
[link]:  http://path.to/link.html "Some Link" class=external
         style="border: solid black 1px;"
```

This will generate width and height attributes for the image, and a border around the link. And while it can be argued that it does look "like it's been marked up with tags [and] formatting instructions", even I can't argue too strongly against it. The link and the title in quotes already look like some form of markup, and the the additional tags are hardly that intrusive, and they offer a great deal

of functionality. They might even be useful in further functions (citations?).

The attributes must continue after the other link/image data, and may contain newlines, but must start at the beginning of the line. The format is `attribute=value` or `attribute="multi word value"`. Currently, MultiMarkdown does not attempt to interpret or make any use of any of these attributes. Also, you can't have a multiword attribute span a newline.

Images

The basic syntax for images in Markdown is:

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

![Alt text](/path/to/img.jpg "Optional title")


![Alt text][id]

[id]: url/to/image  "Optional title attribute"
```

In addition to the attributes you can use with links and images (described in the previous section), MultiMarkdown also adds a few additional things. If an image is the only thing in a paragraph, it is treated as a block level element:

```
This image (![Alt text](/path/to/img.jpg))
is different than the following image:

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

The resulting HTML is:

```
<p>This image (<img src="/path/to/img.jpg" alt="Alt text" />)
is different than the following image:</p>

<figure>
<img src="/path/to/img.jpg" alt="Alt text" />
<figcaption>Alt text</figcaption>
</figure>
```

The first one would be an inline image. The second one (in HTML) would be wrapped in an HTML `figure` element. In this case, the `alt` text is also used as a figure caption, and can contain MultiMarkdown syntax (e.g. bold, emph, etc.). The alt text is not specifically designed to limit which MultiMarkdown is supported, but there will be limits and block level elements aren't supported.

Tables

Table Basics

MultiMarkdown has a special syntax for creating tables. It is generally compatible with the syntax used by Michael Fortin for PHP Markdown Extra[47]

[47] http://www.michelf.com/projects/php-markdown/extra/

Basically, it allows you to turn:

```
|             |          Grouping           ||
First Header  | Second Header | Third Header |
 ------------ | :-----------: | -----------: |
Content       |          *Long Cell*        ||
Content       |   **Cell**    |         Cell |

New section   |     More      |         Data |
And more      | With an escaped '\|'         ||
[Prototype table]
```

into the following table (subsection 1).

Table 1: Prototype table

	Grouping	
First Header	Second Header	Third Header
Content	*Long Cell*	
Content	**Cell**	Cell
New section	More	Data
And more	With an escaped '|'	

Table Rules

The requirements are:

- There must be at least one `|` per line
- The "separator" line between headers and table content must contain only `|`, `-`, `=`, `:`, `.`, `+`, or spaces
- Cell content must be on one line only

- Columns are separated by `|`
- The first line of the table, and the alignment/divider line, must start at the beginning of the line

  Other notes:

- It is optional whether you have `|` characters at the beginning and end of lines.
- The "separator" line uses `----` or `====` to indicate the line between a header and cell. The length of the line doesn't matter, but must have at least one character per cell.
- To set alignment, you can use a colon to designate left or right alignment, or a colon at each end to designate center alignment, as above. If no colon is present, the default alignment of your system is selected (left in most cases). If the separator line ends with `+`, then cells in that column will be wrapped when exporting to LaTeX if they are long enough.
- To indicate that a cell should span multiple columns, then simply add additional pipes (`|`) at the end of the cell, as shown in the example. If the cell in question is at the end of the row, then of course that means that pipes are not optional at the end of that row.... The number of pipes equals the number of columns the cell should span.
- You can use normal Markdown markup within the table cells.
- Captions are optional, but if present must be at the beginning of the line immediately preceding or following the table, start with `[`, and end with `]`. If you have a caption before and after the table, only the first match will be used.
- If you have a caption, you can also have a label, allowing you to create anchors pointing to the table. If there is no label, then the caption acts as the label
- Cells can be empty.
- You can create multiple `<tbody>` tags (for HTML) within a table by having a **single** empty line between rows of the table. This allows your CSS to place horizontal borders to emphasize different sections of the table. This feature doesn't work in all output formats (e.g. RTF and OpenDocument).

Limitations of Tables

- MultiMarkdown table support is designed to handle *most* tables for *most* people; it doesn't cover *all* tables for *all* people. If you need complex tables you will need to create them by hand or with a tool specifically designed for your output format. At some point, however, you should consider whether a table is really the best approach if you find MultiMarkdown tables too limiting.
- Native RTF support for tables is *very* limited. If you need more complex tables, I recommend using the OpenDocument format, and then using LibreOffice[48] to convert your document to RTF.

[48] http://www.libreoffice.org/

Footnotes

I have added support for footnotes to MultiMarkdown, using the syntax proposed by John Gruber. Note that there is no official support for footnotes yet, so the output format may change, but the input format sounds fairly stable.

To create a footnote, enter something like the following:

```
Here is some text containing a footnote.[^somesamplefootnote]

[^somesamplefootnote]: Here is the text of the footnote itself.

[somelink]:http://somelink.com
```

The footnote itself must be at the start of a line, just like links by reference. If you want a footnote to have multiple paragraphs, lists, etc., then the subsequent paragraphs need an extra tab preceding them. You may have to experiment to get this just right, and please let me know of any issues you find.

This is what the final result looks like:

Here is some text containing a footnote.[49]

[49] Here is the text of the footnote itself.

You can also use "inline footnotes":

```
Here is another footnote.[^This is the footnote itself]
```

Citations

I have included support for *basic* bibliography features in this version of MultiMarkdown. Please give me feedback on ways to improve this but keep the following in mind:

1. Bibliography support in MultiMarkdown is rudimentary. The goal is to offer a basic standalone feature, that can be changed using the tool of your choice to a more robust format (e.g. BibTeX, CiteProc). My XSLT files demonstrate how to make this format compatible with BibTeX, but I am not planning on personally providing compatibility with other tools. Feel free to post your ideas and tools to the wiki.

2. Those needing more detailed function sets for their bibliographies may need customized tools to provide those services. This is a basic tool that should work for most people. Reference librarians will probably not be satisfied however.

To use citations in MultiMarkdown, you use a syntax much like that for anchors:

```
This is a statement that should be attributed to
its source[p. 23][#Doe:2006].

And following is the description of the reference to be
used in the bibliography.

[#Doe:2006]: John Doe. *Some Big Fancy Book*.  Vanity Press, 2006.
```

In HTML output, citations are indistinguishable from footnotes.

You are not required to use a locator (e.g. p. 23), and there are no special rules on what can be used as a locator if you choose to use one. If you prefer to omit the locator, just use an empty set of square brackets before the citation:

```
This is a statement that should be attributed to its
source[][#Doe:2006].
```

There are no rules on the citation key format that you use (e.g. `Doe:2006`), but it must be preceded by a `#`, just like footnotes use `^`.

As for the reference description, you can use Markup code within this section, and I recommend leaving a blank line afterwards to prevent concatenation of several references. Note that there is no way to reformat these references in different bibliography styles; for this you need a program designed for that purpose (e.g. BibTeX).

If you want to include a source in your bibliography that was not cited, you may use the following:

```
[Not cited][#citekey]
```

The `Not cited` bit is not case sensitive.

If you are creating a LaTeX document, the citations will be included, and natbib will be used by default. If you are not using BibTeX and are getting errors about your citations not being compatible with 'Author-Year', you can add the following to your documents metadata:

```
latex input:        mmd-natbib-plain
```

This changes the citation style in natbib to avoid these errors, and is useful when you include your citations in the MultiMarkdown document itself.

BibTeX

If you are creating a LaTeX document, and need a bibliography, then you should definitely look into BibTeX[50] and natbib[51]. It is beyond the scope of this document to describe how these two packages work, but it is possible to combine them with MultiMarkdown.

[50] http://www.bibtex.org/

[51] http://merkel.zoneo.net/Latex/natbib.php

To use BibTeX in a MultiMarkdown document, you need to use the `BibTeX` metadata (section) to specify where your citations are stored.

Since `natbib` is enabled by default, you have a choice between using the `\citep` and `\citet` commands. The following shows how this relates to the MultiMarkdown syntax used.

```
[#citekey]    => ~\citep{citekey}
[#citekey][]  => ~\citep{citekey}

[foo][#citekey] => ~\citep[foo]{citekey}

[foo\]\[bar][#citekey] => ~\citep[foo][bar]{citekey}


[#citekey;]    => \citet{citekey}
[#citekey;][]  => \citet{citekey}

[foo][#citekey;] => \citet[foo]{citekey}

[foo\]\[bar][#citekey;] => \citet[foo][bar]{citekey}
```

Definition Lists

MultiMarkdown has support for definition lists using the same syntax used in PHP Markdown Extra[52]. Specifically:

[52] http://www.michelf.com/projects/php-markdown/extra/

```
Apple
:   Pomaceous fruit of plants of the genus Malus in
    the family Rosaceae.
:   An american computer company.

Orange
:   The fruit of an evergreen tree of the genus Citrus.
```

becomes:

Apple Pomaceous fruit of plants of the genus Malus in the family Rosaceae.

An american computer company.

Orange The fruit of an evergreen tree of the genus Citrus.

You can have more than one term per definition by placing each term on a separate line. Each definition starts with a colon, and you can have more than one definition per term. You may optionally have a blank line between the last term and the first definition.

Definitions may contain other block level elements, such as lists, blockquotes, or other definition lists.

Unlike PHP Markdown Extra, all definitions are wrapped in <p> tags. First, I was unable to get Markdown *not* to create paragraphs. Second, I didn't see where it mattered - the only difference seems to be aesthetic, and I actually prefer the <p> tags in place. Let me know if this is a problem.

See the PHP Markdown Extra[53] page for more information.

[53] http://www.michelf.com/projects/php-markdown/extra/

Abbreviations

MultiMarkdown includes support for abbreviations, as implemented in Michel Fortin's PHP Markdown Extra[54]. Basically, you define an abbreviation using the following syntax:

[54] http://michelf.ca/projects/php-markdown/extra/

```
*[HTML]: HyperText Markup Language
*[W3C]:  World Wide Web Consortium
```

Then, wherever you use the words `HTML` or `W3C` in your document, the `abbr` markup will be added:

```
The HTML specification
is maintained by the W3C.
```

becomes:

```
The <abbr title="Hyper Text Markup Language">HTML</abbr> specification
is maintained by the <abbr title="World Wide Web Consortium">W3C</abbr>.
```

Here's an example using HTML and World Wide Web Consortium (W3C). The exact behavior will depend on which format you are viewing this document in. Especially if we use HTML and W3C again. (Remember that HTML has probably already been used if you're viewing a longer version of this document.)

As in PHP Markdown Extra, abbreviations are case-sensitive and will work on multiple word abbreviations. In this case, MultiMarkdown is tolerant of different variations of whitespace between words.

```
Operation Tigra Genesis is going well.

*[Tigra Genesis]:
```

An abbreviation with an empty definition results in an omitted `title` attribute.

There are a few limitations:

- The full name of the abbreviation is plain text only – no MultiMarkdown markup will be processed.
- Abbreviations don't do anything when exporting to ODF – there's not an equivalent structure there – it would have to be hand coded. I may or may not get around to this, but pull requests welcome. ;)
- When exporting to LaTeX, the `acronym` package is used; this means that the first usage will result in `full text (short)`, and subsequent uses will result in `short`.

Fenced Code Blocks

In addition to the regular indented code block that Markdown uses, you can use "fenced" code blocks in MultiMarkdown. These code blocks do not have to be indented, and can also be configured to be compatible with a third party syntax highlighter. These code blocks should begin with 3 to 5 backticks, an optional language specifier (if

using a syntax highlighter), and should end with the same number of backticks you started with:

```
# Demonstrate Syntax Highlighting if you link to highlight.js #
# http://softwaremaniacs.org/soft/highlight/en/
print "Hello, world!\n";
$a = 0;
while ($a < 10) {
print "$a...\n";
$a++;
}
```

I don't recommend any specific syntax highlighter, but have used the following metadata to set things up. It may or may not work for you:

```
html header:    <link rel="stylesheet" href="http://yandex.st/highlightjs/7.3/styles/default.min.css"
    <script src="http://yandex.st/highlightjs/7.3/highlight.min.js"></script>
    <script>hljs.initHighlightingOnLoad();</script>
```

Fenced code blocks are particularly useful when included another file (File Transclusion (section)), and you want to show the *source* of the file, not what the file looks like when processed by MultiMarkdown.

Math

MultiMarkdown 2.0 used ASCIIMathML[55] to typeset mathematical equations. There were benefits to using ASCIIMathML, but also some disadvantages.

When rewriting for MultiMarkdown 3.0, there was no straightforward way to implement ASCIIMathML which lead me to look for alternatives. I settled on using MathJax[56]. The advantage here is that the same syntax is supported by MathJax in browsers, and in LaTeX.

This does mean that math will need to be entered into MultiMarkdown documents using the LaTeX syntax, rather than ASCIIMathML.

To enable MathJax support in web pages, you have to include a link to an active MathJax installation — setting this up is beyond the scope of this document, but it's not too hard.

Here's an example of the metadata setup, and some math:

[55] http://www1.chapman.edu/~jipsen/mathml/asciimath.html

[56] http://www.mathjax.org/

```
latex input:    mmd-article-header
Title:          MultiMarkdown Math Example
latex input:    mmd-article-begin-doc
latex footer:   mmd-memoir-footer
```

```
HTML header:    <script type="text/javascript"
                src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML">
                </script>


An example of math within a paragraph --- \\({e}^{i\pi }+1=0\\)
--- easy enough.

And an equation on it's own:

\\[ {x}_{1,2}=\frac{-b\pm \sqrt{{b}^{2}-4ac}}{2a} \\]

That's it.
```

Here's what it looks like in action (if you're viewing this document in a supported format):

An example of math within a paragraph — $e^{i\pi}+1=0$ — easy enough.

And an equation on it's own:

$$x_{1,2}=\frac{-b\pm \sqrt{b^2-4ac}}{2a}$$

That's it.

In addition to the `\\[ \\]` and `\\( \\)` syntax, you can use LaTeX style "dollar sign" delimiters:

```
An example of math within a paragraph --- ${e}^{i\pi }+1=0$
--- easy enough.

And an equation on it's own:

$${x}_{1,2}=\frac{-b\pm \sqrt{{b}^{2}-4ac}}{2a}$$

That's it.
```

In order to be correctly parsed as math, there must not be any space between the `$` and the actual math on the inside of the delimiter, and there *must* be space on the outside.

Superscripts and Subscripts

You can easily include superscripts and subscripts in MultiMarkdown as well:

```
This apartment has an area of 100m^2
One must consider the value of x~z
```

becomes

This apartment has an area of 100m^2
One must consider the value of x_z

The subscript must not contain any whitespace or punctuation.

More complicated exponents and subscripts can be delimited like this:

```
y^(a+b)^
x~y,z~
```

$y^{(a+b)}$
$x_{y,z}$

Glossaries

MultiMarkdown has a feature that allows footnotes to be specified as glossary terms. It doesn't do much for XHTML documents, but the XSLT file that converts the document into LaTeX is designed to convert these special footnotes into glossary entries.

The glossary format for the footnotes is:

```
[^glossaryfootnote]: glossary: term (optional sort key)
    The actual definition belongs on a new line, and can continue on
    just as other footnotes.
```

The `term` is the item that belongs in the glossary. The `sort key` is optional, and is used to specify that the term should appear somewhere else in the glossary (which is sorted in alphabetical order).

Unfortunately, it takes an extra step to generate the glossary when creating a pdf from a latex file:

1. You need to have the `basic.gst` file installed, which comes with the memoir class.
2. You need to run a special makeindex command to generate the `.glo` file: `makeindex -s 'kpsewhich basic.gst' -o "filename.gls" "filename.glo"`
3. Then you run the usual pdflatex command again a few times.

Alternatively, you can use the code below to create an engine file for TeXShop (it belongs in ~/Library/TeXShop/Engines). You can name it something like MemoirGlossary.engine. Then, when processing a file that needs a glossary, you typeset your document once with this engine, and then continue to process it normally with the usual LaTeX engine. Your glossary should be compiled appropriately. If you use TeXShop[57], this is the way to go.

[57] http://www.uoregon.edu/~koch/texshop/

Note: *Getting glossaries to work is a slightly more advanced LaTeX feature, and might take some trial and error the first few times.*

```
#!/bin/

set path = ($path /usr/local/teTeX/bin/powerpc-apple-darwin-current
    /usr/local/bin) # This is actually a continuation of the line above

set basefile = ‘basename "$1" .tex‘

makeindex -s ‘kpsewhich basic.gst‘ -o "${basefile}.gls" "${basefile}.glo"
```

CriticMarkup

What Is CriticMarkup?

> CriticMarkup is a way for authors and editors to track changes to documents in plain text. As with Markdown, small groups of distinctive characters allow you to highlight insertions, deletions, substitutions and comments, all without the overhead of heavy, proprietary office suites. http://criticmarkup.com/

CriticMarkup is integrated with MultiMarkdown itself, as well as MultiMarkdown Composer[58]. I encourage you to check out the web site to learn more as it can be a very useful tool. There is also a great video showing CriticMarkup in use while editing a document in MultiMarkdown Composer.

[58] http://multimarkdown.com/

The CriticMarkup Syntax

The CriticMarkup syntax is fairly straightforward. The key thing to remember is that CriticMarkup is processed *before* any other MultiMarkdown is handled. It's almost like a separate layer on top of the MultiMarkdown syntax.

When editing in MultiMarkdown Composer, you can have CriticMarkup syntax flagged in the both the editor pane and the preview window. This will allow you to see changes in the HTML preview.

When using CriticMarkup with MultiMarkdown itself, you have four choices:

- Leave the CriticMarkup syntax in place (`multimarkdown foo.txt`)
- Accept all changes, giving you the "new" document (`multimarkdown -a foo.txt`)
- Reject all changes, giving you the "original" document (`multimarkdown -r foo.txt`)
- Attempt to show the changes as highlights. This only works in HTML, and to use it you ask for the new and original document at the same time (`multimarkdown -a -r foo.txt`)

CriticMarkup comments and highlighting are ignored when processing.

Deletions from the original text:

```
This is {--is --}a test.
```

Additions:

```
This {++is ++}a test.
```

Substitutions:

```
This {~~isn't~>is~~} a test.
```

Highlighting:

```
This is a {==test==}.
```

Comments:

```
This is a test{>>What is it a test of?<<}.
```

My philosophy on CriticMarkup

I view CriticMarkup as two things:

1. A syntax for documenting editing notes and changes, and for collaborating amongst coauthors.
2. A means to display those notes/changes in the HTML output.

I believe that #1 is a really great idea, and well implemented. #2 is not so well implemented, largely due to the "orthogonal" nature of CriticMarkup and the underlying Markdown syntax.

CM is designed as a separate layer on top of Markdown/MultiMarkdown. This means that a Markdown span could, for example, start in the middle of a CriticMarkup structure, but end outside of it. This means that an algorithm to properly convert a CM/Markdown document to HTML would be quite complex, with a huge number of edge cases to consider. I've tried a few (fairly creative, in my opinion) approaches, but they didn't work. Perhaps someone else will come up with a better solution, or will be so interested that they put the work in to create the complex algorithm. I have no current plans to do so.

Additionally, there is a philosophical distinction between documenting editing notes, and using those notes to produce a "finished" document (e.g. HTML or PDF) that keeps those editing notes intact (e.g. strikethroughs, highlighting, etc.) I believe that CM is incredibly useful for the editing process, but am less convinced for the output process (I know many others disagree with me, and that's ok. And to be clear, I think that what Gabe and Erik have done with CriticMarkup is fantastic!)

There are other CriticMarkup tools besides MultiMarkdown and MultiMarkdown Composer[59], and you are more than welcome to use them.

[59] http://multimarkdown.com/

For now, the *official* MultiMarkdown support for CriticMarkup consists of:

1. CriticMarkup syntax is "understood" by the MultiMarkdown parser, and by MultiMarkdown Composer syntax highlighting.

2. When converting from MultiMarkdown text to an output format, you can ignore CM formatting (probably not what you want to do), accept all changes, or reject all changes (as above). These are the preferred choices.

3. The secondary to choice, when exporting to HTML, is to *attempt* to show the changes in the HTML output. Because the syntaxes are orthogonal, this will not always work properly, and will not always give valid HTML output.

Raw HTML

You can include raw (X)HTML within your document. Exactly what happens with these portions depends on the output format. You can also use the `markdown` attribute to indicate that MultiMarkdown processing should be applied within the block level HTML tag. This is

in addition to the `--process-html` command line option that causes MultiMarkdown processing to occur within *all* block level HTML tags.

For example:

```
<div>This is *not* MultiMarkdown</div>

<div markdown=1>This *is* MultiMarkdown</div>
```

will produce the following without `--process-html`:

```
<div>This is *not* MultiMarkdown</div>

<div>This <em>is</em> MultiMarkdown</div>
```

and with `--process-html`:

```
<div>This is <em>not</em> MultiMarkdown</div>

<div>This <em>is</em> MultiMarkdown</div>
```

However, the results may be different than anticipated when outputting to LaTeX or other formats. Normally, block level HTML will be ignored when outputting to LaTeX or ODF. The example above would produce the following, leaving out the first <`div`> entirely:

```
This \emph{is} MultiMarkdown
```

And this with `--process-html`:

```
This is \emph{not} MultiMarkdown
This \emph{is} MultiMarkdown
```

You will also notice that the line breaks are different when outputting to LaTeX or ODF, and this can cause the contents of two <`div`> tags to be placed into a single paragraph.

Raw LaTeX/OpenDocument/etc.

You can use HTML comments to include additional text that will be included in the exported file without being changed. This can be used for any export format, which means that each document can only be configured for one export format at a time. In other

words, it is highly unlikely that valid raw LaTeX will also be valid OpenDocument source code.

```
This will be processed by *MultiMarkdown*.
<!-- This will not be processed by *MultiMarkdown -->
```

File Transclusion

File transclusion is the ability to tell MultiMarkdown to insert the contents of another file inside the current file being processed. For example:

```
This is some text.

{{some_other_file.txt}}

Another paragraph
```

If a file named `some_other_file.txt` exists, its contents will be inserted inside of this document *before* being processed by MultiMarkdown. This means that the contents of the file can also contain MultiMarkdown formatted text.

If you want to display the *contents* of the file without processing it, you can include it in a code block (you may need to remove trailing newlines at the end of the document to be included):

```
This is some text

‘‘‘
{{relative/path/to/some_other_file.txt}}
‘‘‘

Another paragraph
```

Transclusion is recursive, so the file being inserted will be scanned to see if it references any other files.

Metadata in the file being inserted will be ignored. This means that the file can contain certain metadata when viewed alone that will not be included when the file is transcluded by another file.

You can use the Transclude Base (subsection) metadata to specify where MultiMarkdown should look for the files to be included. All files must be in this folder. If this folder is not specified, then Multi-

Markdown will look in the same folder as the parent file.

Note: Thanks to David Richards for his ideas in developing support for this feature.

Wildcard Extensions

Sometimes you may wish to transclude alternate versions of a file depending on your output format. Simply use the extension ".*" to have MMD choose the proper version of the file (e.g. `foo.tex`, `foo.fodt`, `foo.html`, etc.)

```
Insert a different version of a file here based on export format:
{{foo.*}}
```

Escaped newlines

Thanks to a contribution from Nicolas[60], MultiMarkdown has an additional syntax to indicate a line break. The usual approach for Markdown is "space-space-newline" — two spaces at the end of the line. For some users, this causes problems:

[60] https://github.com/njmsdk

- the trailing spaces are typically invisible when glancing at the source, making it easy to overlook them
- some users' text editors modify trailing space (IMHO, the proper fix for this is a new text editor. . .)

Nicolas submitted a patch that enables a new option that interprets "\" before a newline as a marker that a line break should be used:

```
This is a line.\
This is a new line.
```

To enable this feature, use the following option:

```
multimarkdown --escaped-line-breaks file.txt
```

If this option is not enabled, then the default behavior will be to treat the newline as an escaped character, which results in it simply appearing as a newline character in the output. This means that the default behavior is the same as if the "\" is not in the source file.

Known Issues

OpenDocument

OpenDocument doesn't properly support image dimensions

It's relatively easy to insert an image into ODF using fixed dimensions, but harder to get a scaled image without knowing the exact aspect ratio of the image.

For example, in LaTeX or HTML, one can specify that image should be scaled to 50% of the width, and have it automatically calculate the proper height. This does not work in ODF, at least not that I can find.

You have to manually adjust the image to fit your desired constraint. It's easy to do, simply hold down the shift key while adjusting the image size, and it will likely snap to match the specified dimension.

I welcome suggestions on a better way to do this.

RTF

- Non-ASCII characters are not supported
- Lists are not proper lists
- Images are not supported
- Tables are not fully supported

OPML

OPML doesn't handle "skipped" levels

When converting a MMD text file to OPML with the mmd binary, each level only contains it's direct children. For example:

```
# First Level #
```

```
## Second Level  ##

### Third Level ###

## Another Second Level ##

#### Fourth Level ####
```

When this is converted to OPML, the "Fourth Level" item will be deleted, since it skips a level from its parent, "Another Second Level".

It's possible to fix this, but it's going to take a more complicated algorithm than what I currently have and it's not a high priority for me to fix at the moment.

As always, suggestions welcome.

Things Yet to Be Done

RTF

- Support lists
- Improve table support
- Support Non-ASCII characters
- Code span

More Information

For more information about MultiMarkdown, visit the following sites:

- http://fletcherpenney.net/multimarkdown/
- https://github.com/fletcher/MultiMarkdown-4/
- https://groups.google.com/forum/#!forum/multimarkdown

Acknowledgements

Thanks to the individuals and groups below for their contributions to improving Markdown and MultiMarkdown:

- John Gruber
- Michel Fortin
- Jonathan Weber
- Mark Eli Kalderon
- Choan C. Gálvez
- Dr. Drang
- Robert McGonegal
- David Green
- Trey Pickard
- Saleem
- Melinda Norris
- Sean Wallace
- Allan Odgaard
- Stefan Brantschen
- Keith Blount
- Gerd Knops
- John Purnell
- Jonathan Coulombe (special thanks for helping troubleshoot MMD 3.0!)
- Jason Bandlow
- Joakim Hertze

- Kee-Lin Steven Chan
- Vasil Yaroshevich
- Matt Neuburg
- James Howison
- Edward Nixon
- etherean
- Özgür Gökmen
- Chad Schmidt
- Greg (gr)
- Ben Jennings
- Silvan Kaiser
- Tomas Doran
- Rob Walton
- Dan Rolander
- Duoyi wu
- Dan Dascalescu
- Ingolf Schäfer
- Chris Bunch
- Oblomov
- Alex Melhuish
- Stephan Mueller
- Josh Brown
- Rob Person
- Matthew D. Rankin
- Dawid Ciężarkiewicz
- Joonas Pulakka
- ipetraka
- John MacFarlane (special thanks for creating peg-markdown[61] and helping me to get started on MMD 3.0!)

[61] https://github.com/jgm/peg-markdown

- David Sparks
- Katie Floyd
- Daniel Müller
- Daniel Jalkut (special thanks for helping to remove glib dependency!)
- Jon Skovron
- Jake Walker
- Michael Heilemann
- Brett Terpstra
- Charles Cowan
- David Richards
- Thomas Hodgson

and others I have surely forgotten. . . .