MultiMarkdown User's Guide
Version 3.0

Fletcher T. Penney
http://fletcherpenney.net/mmd/

May 18, 2011

Contents

Chapter 1

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]

1.1 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:

> 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]

1.2 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,

[1] http://MacSparky.com/
[2] http://daringfireball.net/projects/markdown/
[3] http://daringfireball.net/projects/markdown/

including complete 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.

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.

1.3 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:

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

```
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.

1.4 Where is this Guide Kept?

This User's Guide is designed to be viewed as a single HTML document, or as a PDF after being processed by LaTeX. When processed by MultiMarkdown and viewed as a single document, it will demonstrate many of the features that MultiMarkdown offers, and should be fairly easy to navigate.

The source text documents are actually stored in a wiki on Github so that the document can easily be worked on by anyone with a Github account:

- https://github.com/fletcher/peg-multimarkdown/wiki/User's-Manual

Note: Currently, the Github wiki software supports Markdown, but not MultiMarkdown. This means that certain MultiMarkdown specific features will not work on the wiki, but will work when the source is processed by MultiMarkdown separately from the wiki.

You can download a copy of the source files using the instructions contained at the wiki itself. Additionally, the source is linked to from within the source project for MultiMarkdown in the `documentation` directory. If you download the source, you can can compile a copy of the documentation automatically with:

```
make docs
```

(This command requires perl to be installed — maybe at some point I'll work on an alternative for Windows users without perl).

Chapter 2

How do I install MultiMarkdown?

The "Basic" instructions below will get you a functioning installation of MultiMarkdown, including the following commands:

- `multimarkdown`
- `mmd`
- `mmd2tex`
- `mmd2opml`
- `mmd2odf`

The "Advanced" instructions will help if you want to make use of some of the more advanced features, such as custom XSLT output, you will need to install the MultiMarkdown Support files.

If you want to use the LaTeX features, it would be helpful to install the LaTeX support files.

The instructions are provided for several operating systems:

- Mac OS X (section 2.1)
- Windows (section 2.2)
- Linux (section 2.3)

2.1 Mac OS X

Basic

On the Mac, you can choose from using an installer to install the program and required libraries for you, or you can compile it yourself from scratch. If you know what that means, follow the instructions below in the Linux section. Otherwise, definitely go for the installer!

1. Download the latest version of the `MultiMarkdown-Mac` installer from the downloads[1] page
2. Unzip the downloaded file, then run the installer and follow the onscreen directions.

[1] http://github.com/fletcher/peg-multimarkdown/downloads

Advanced

If you want to use XSLT[2] to customize your output, or you want to use MultiMarkdown with Scrivener[3] or TextMate[4], you should also download and run the `MultiMarkdown-Support-Mac` installer[5].

If you want to use LaTeX to generate PDFs, I recommend downloading the LaTeX support files[6] and placing them in `~/Library/texmf/tex/latex/mmd`.

Compiling

The instructions for compiling on Mac OS X are similar to the Linux (section 2.3) instructions below. If you want to build an installer, you can switch to the `mac-installer` branch, and use the `make mac-installer` command.

On Mac OS X, please note that you will need to install the Developer Tools. You then need to install a few libraries. I used homebrew[7]:

```
brew install pkg-config glib gettext
```

You can also use fink[8] or something similar (though this caused some machine architecture incompatibilities for me):

```
fink install glib2-shlibs glib2-dev
```

2.2 Windows

Basic

The easiest way to install MMD on Windows is the `MultiMarkdown-Windows` installer from the downloads[9] page and run it. The installer is built using software by BitRock[10].

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.

Advanced

If you intend to use LaTeX to generate PDFs, you will need to install a working version of LaTeX. I have used TeX Live[11] in the past, but the details of installing that are up to you. I would also recommend installing the LaTeX support files[12].

[2] http://en.wikipedia.org/wiki/XSLT
[3] http://www.literatureandlatte.com/scrivener.html
[4] http://macromates.com/
[5] http://github.com/fletcher/peg-multimarkdown/downloads
[6] https://github.com/fletcher/peg-multimarkdown-latex-support
[7] https://github.com/mxcl/homebrew
[8] http://www.finkproject.org/
[9] http://github.com/fletcher/peg-multimarkdown/downloads
[10] http://bitrock.com/
[11] http://www.tug.org/texlive/
[12] https://github.com/fletcher/peg-multimarkdown-latex-support

If you want to use XSLT[13] to customize your output, you will need to install a working version of `xsltproc` and a program to allow you to run shell scripts so that you can use the MMD-Support[14] utility scripts. This is fairly complicated to install on Windows, so I highly recommend that you determine whether you really need these features. Most of the customizations you need can actually be done within the LaTeX include files specified in the metadata, without requiring XSLT. Check out some of the examples in the MultiMarkdown Gallery[15] and in the LaTeX Support files for ideas of what can be done.

Compiling

If you want to compile this yourself, you do it in the same way that you would install peg-markdown for Windows. The instructions are on the peg-multimarkdown wiki[16]. I was able to compile for Windows fairly easily using Ubuntu linux following those instructions. I have not tried to actually compile on a Windows machine.

Otherwise, see the section on Linux (section 2.3) for further tips.

(For those who are interested, I actually created an EC2 instance on Amazon and installed the necessary software to compile. It was pretty easy and probably cost me $0.02...)

2.3 Linux

Basic compiling

Currently the only way to install MultiMarkdown on *nix machines is to compile it yourself. You need to have `glib2` installed. For example, on ubuntu:

```
sudo apt-get install libglib2.0-dev
```

For other linux distributions, you will need to do something similar.

Once you have the libraries installed, you can either download the source from the downloads[17] page, or (preferentially) you can use git:

```
git clone --recursive git://github.com/fletcher/peg-multimarkdown.git
```

At this point you can compile with:

```
make
```

Advanced compiling

One additional step to include if you use git to clone the source, is to run the `update_submodules.sh` script. This script basically runs two commands, that you can also run by hand:

[13] http://en.wikipedia.org/wiki/XSLT
[14] https://github.com/fletcher/MMD-Support
[15] https://github.com/fletcher/MultiMarkdown-Gallery
[16] https://github.com/fletcher/peg-multimarkdown/wiki/Building-for-Windows
[17] http://github.com/fletcher/peg-multimarkdown/downloads

```
git submodule init
git submodule update
```

Then, simply run `make` to compile the source. You can also run some test commands to verify that everything is working properly. Of note, it is normal to fail one test in the Markdown tests, but the others should pass. You can then install the binary and utility scripts wherever you like, though for consistency `/usr/local/bin` would probably be best.

```
make
make test
make mmdtest
make latextest
```

As above, if you intend to use XSLT[18], LaTeX[19], and the like, I recommend also installing the LaTeX Support files[20] and MultiMarkdown Support[21].

[18] http://en.wikipedia.org/wiki/XSLT
[19] http://en.wikipedia.org/wiki/LaTeX
[20] https://github.com/fletcher/peg-multimarkdown-latex-support
[21] https://github.com/fletcher/MMD-Support

Chapter 3

How do I 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.

3.1 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 OPML:

```
multimarkdown -t opml 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 -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
```

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 -x, multimarkdown --extensions
```

This turns on all extensions. It is really intended to be used when you would like to use some of the extensions of peg-markdown. Probably not useful for MultiMarkdown users, but I left it turned on just in case, except where it interferes with the desired MMD functionality.

```
multimarkdown --filter-html
multimarkdown --filter-styles
```

The first command causes any raw HTML (except styles) included in the source document to *not* be included in the output. The second command removes HTML style tags from the output.

```
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 -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 -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 --smart
```

Uses smart typography, similar to John Gruber's SmartyPants[1] program, which was included in MultiMarkdown 2.0. This extension is turned on by default in MultiMarkdown.

```
multimarkdown --notes
```

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

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

This option tells MultiMarkdown to process the text included within HTML tags in the source document.

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

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

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.

3.2 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 TextMate

If you want to run MultiMarkdown from directly within TextMate[2], you should install my MultiMarkdown bundle[3]. This is a modified version of the original Markdown bundle for TextMate that includes better support for MultiMarkdown.

This bundle will work with MultiMarkdown 2.0, or with MultiMarkdown 3.0 if you install the MultiMarkdown-Support files (available from the downloads page[4]).

Using MultiMarkdown with Scrivener

Scrivener[5] is a great program for writers using Mac OS X. It includes built in support for MultiMarkdown. If you want to use MultiMarkdown 3.0 with Scrivener, you need to install the Support files in `~/Library/Application Support/MultiMarkdown`. The MultiMarkdown-Support installer is available from the downloads page[6] and will install these 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 downloads[7] page, or by running `make drop` from the command line in the `multimarkdown` source directory.

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 two Quick Look generators that allow the finder to preview the contents of a MultiMarkdown text file as an HTML preview. One works on plain text files, and one works on OPML files.

Available for download from github[8].

3.3 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.

[2] http://macromates.com/
[3] https://github.com/fletcher/markdown.tmbundle
[4] http://github.com/fletcher/peg-multimarkdown/downloads
[5] http://www.literatureandlatte.com/scrivener.html
[6] http://github.com/fletcher/peg-multimarkdown/downloads
[7] http://github.com/fletcher/peg-multimarkdown/downloads
[8] https://github.com/fletcher/qlmultimarkdown/downloads

3.4 MultiMarkdown and LaTeX

Of note LaTeX[9] 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.

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[10], 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 (section 9.5).

3.5 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[11] is a fabulous outliner that supports this field. If you're into mind mapping software,

[9] http://en.wikipedia.org/wiki/LaTeX
[10] http://groups.google.com/group/multimarkdown/
[11] http://www.omnigroup.com/applications/omnioutliner/

iThoughts[12] 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[13] package:

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

3.6 MultiMarkdown and OpenDocument

It is also possible to convert a MultiMarkdown text file into a word processing document for OpenOffice.org[14] or LibreOffice[15]. 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...)

```
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[16] 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.

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.

[12] http://www.ithoughts.co.uk/
[13] https://github.com/fletcher/MMD-Support
[14] http://www.openoffice.org/
[15] http://www.libreoffice.org/download
[16] https://github.com/fletcher/peg-multimarkdown/downloads

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

Chapter 4

How do I create a MultiMarkdown document?

The general concept in MultiMarkdown is that it should be easy for someone to type a plain text file that is human-readable, and then use the MultiMarkdown program to convert that text file into a more complicated computer language such as HTML or LaTeX. This allows you to create high quality output without having to spend hours and hours fiddling with font sizes, margins, etc.

The first step in learning to use MultiMarkdown is to learn how to use Markdown[1]. MultiMarkdown is an extension to Markdown, and builds off of the basic fundamentals used in Markdown.

I recommend starting by familiarizing yourself with the Markdown basics[2] and syntax[3] pages.

Once you're familiar with the basics of Markdown, it will be relatively easy to pick up the advanced features included in MultiMarkdown.

4.1 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.

```
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.
Test:    And this is a new key-value pair
```

The syntax for including metadata is simple.

[1] http://daringfireball.net/projects/markdown/
[2] http://daringfireball.net/projects/markdown/basics
[3] http://daringfireball.net/projects/markdown/syntax

- The metadata must begin at the very top of the document - no blank lines can precede it.
- 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 a letter or number, then the following characters can consist of 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 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.

4.2 Complete documents vs "snippets"

In order to include metadata information such as a title, the HTML document created by MultiMarkdown must be "complete." This means that it starts with something that looks like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

If you include metadata in your document (with two exceptions), then you will generate a complete document. If you don't include metadata, then you will instead generate a "snippet." The snippet will just include the relevant portion of HTML, but will not include the <`head`> or <`body`> elements.

Metadata that is only intended to affect the way MultiMarkdown processes the output will not trigger a complete document. Currently, this means you can use `Base Header Level` or `Quotes Language` and still output a snippet if you don't include any other metadata.

Document formats other than HTML don't have distinctions between complete documents and snippets. Any included and relevant metadata will be included if present.

4.3 "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 documents to generate the title information.

Affiliation

This is used to enter further information about the author — a link to a website, 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 — i.e. `LaTeX Header Level`, `HTML Header Level`, and `ODF Header Level`.

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
- english
- french
- german
- germanguillemets
- swedish

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 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[4] files 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.

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.

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.

4.4 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 4.1).

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

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

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

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.

4.5 Image and Link 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[5] 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.

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.

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

4.6 Tables

I have implemented a syntax for tables similar to that used by Michael Fortin's PHP Markdown Extra[6].

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      |            And more         ||
[Prototype table]
```

into a table (Table 4.1).

Table 4.1: Prototype table

	Grouping	
First Header	Second Header	Third Header
Content	*Long Cell*	
Content	**Cell**	Cell
New section	More	Data
And more	And more	

The requirements are:

- There must be at least one `|` per line
- The "separator" line 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 `|`'s at the beginning and end of lines.
- 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).

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

- To indicate that a cell should span multiple columns, there 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....
- 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 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.

4.7 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.[7]

4.8 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

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

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.
```

The HTML that is generated is as follows:

```
<p>This is a statement that should be attributed to its source
<span class="markdowncitation"> (<a href="#Doe:2006">1</a>, <span
class="locator">p. 23</span>)</span>.</p>

<p>And following is the description of the reference to be used
in the bibliography.</p>

<div class="bibliography">
<hr />
<p>Bibliography</p>

<div id="Doe:2006"><p>[1] John Doe. <em>Some Big Fancy Book</em>.
Vanity Press, 2006.</p></div>

</div>
```

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.

4.9 BibTeX

If you are creating a LaTeX document, and need a bibliography, then you should definitely look into BibTeX[8] and natbib[9]. It is beyond the scope of this document to describe how these two packages work, but it is possible to combine them with MultiMarkdown.

To use BibTeX in a MultiMarkdown document, you need to use the `BibTeX` metadata (section 4.9) 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}
```

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

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

4.10 Definition Lists

MultiMarkdown has support for definition lists using the same syntax used in PHP Markdown Extra[10]. Specifically:

```
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[11] page for more information.

4.11 Math

MultiMarkdown 2.0 used ASCIIMathML[12] 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[13]. 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:

```
latex input:    mmd-article-header
```

[10] http://www.michelf.com/projects/php-markdown/extra/
[11] http://www.michelf.com/projects/php-markdown/extra/
[12] http://www1.chapman.edu/~jipsen/mathml/asciimath.html
[13] http://www.mathjax.org/

```
Title:          MultiMarkdown Math Example
latex input:    mmd-article-begin-doc
latex footer:   mmd-memoir-footer
HTML header:    <script type="text/javascript"
    src="http://example.net/mathjax/MathJax.js">
    </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.

4.12 Glossary

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[14], this is the way to go.

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"
```

4.13 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>
```

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

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 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 tags to be placed into a single paragraph.

Chapter 5

How do I customize MultiMarkdown Output?

By default, I believe MultiMarkdown does a good job with the default output for the various output formats. That said, it is often desirable to customize the output to your specific needs.

5.1 HTML Customization

The easiest way to customize HTML output is to apply style information using CSS[1]. You can do this by linking to a CSS file with the `CSS` metadata (section 4.3), or by manually including style information in the form of the `HTML Header` metadata (section 4.3).

A more advanced approach is to use XSLT[2] to post-process the HTML output and customize it. You can also write scripts in whatever language you are most familiar with to process the output as desired. You can include these functions in your workflow in whatever method works best for you.

5.2 LaTeX Customization

Similarly, you can customize the appearance of your LaTeX output when using MultiMarkdown. The most straightforward way is through the use of `LaTeX Input` metadata (section 4.3). You can include whatever valid LaTeX code in these files that you desire, and you are limited only by your creativity. You can redefine common commands as needed. You can include various packages, or add default metadata. I recommend examining the samples in the MultiMarkdown Gallery[3] to see some of the customizations I have used so far.

In addition to customizing the output, there are several different "modes" of operation when MultiMarkdown outputs LaTeX. These modes can create various types of documents and rely on common packages. These modes can be specified with the `LaTeX Mode` metadata (section 4.3).

[1] http://en.wikipedia.org/wiki/CSS
[2] http://en.wikipedia.org/wiki/XSLT
[3] https://github.com/fletcher/MultiMarkdown-Gallery

Default LaTeX Mode

The default mode of operation outputs fairly standard LaTeX that should be compatible with most basic document types and packages.

Memoir LaTeX Mode

The `memoir` mode generates output designed to be compatible with the memoir package[4]. This package offers several advantages, and can be used to create books, reports, and other basic document formats.

By default, this User's Guide is processed into a PDF using the memoir class.

Beamer LaTeX Mode

The `beamer` mode generates output designed to be compatible with the beamer package[5]. This package allows you to easily create PDF slideshows (similar to Apple's Keynote or Microsoft's PowerPoint). I used it to create the "What is MultiMarkdown?" slideshow[6].

5.3 OPML Customization

There really isn't anything to customize when creating an OPML document, since it's really just a different way of formatting a MultiMarkdown text document.

5.4 OpenDocument Customization

OpenDocument files use style information in a way that's similar to CSS for HTML. To embed custom style information, you can use the `ODF Header` metadata (section 4.3). This will embed your XML into the header of the document. You'll have to figure out the syntax of what to include on your own, but it's not too difficult if you examine the source of the documents you are interested in emulating.

Since the OpenDocument Flat Text file is an XML file, you could also use XSLT[7] to customize the file as desired.

5.5 Tips and Tricks

There are some creative ways to tailor the output you get based on which format type you are exporting, particularly with HTML and LaTeX. These tricks make use of raw LaTeX (section 3.4), and raw HTML (section 7.7).

Experiment and you can come up with some creative things!

[4] http://www.tex.ac.uk/ctan/macros/latex/contrib/memoir/
[5] https://bitbucket.org/rivanvx/beamer/wiki/Home
[6] https://github.com/fletcher/MultiMarkdown-Gallery/raw/master/What-Is-MMD/what_is_mmd.pdf
[7] http://en.wikipedia.org/wiki/XSLT

Chapter 6

MultiMarkdown Output Formats

MultiMarkdown can output to various formats. The default behavior for each format should be pretty good, but each format does have it's own unique characteristics that sometimes need adjustment.

6.1 HTML

HTML is the "default" format for outputting MultiMarkdown documents, so it's behavior is generally the most predictable.

6.2 LaTeX

LaTeX is the second oldest output format for MultiMarkdown, so many issues have been worked out.

6.3 OpenDocument

OpenDocument Text, or more specifically Flat XML or `.fodt`, is a newer format for MultiMarkdown, so there may still be a few kinks to be worked out.

Images

When outputting to OpenDocument, images are "linked" to in much the same way as when creating HTML. The images are not actually embedded in the document itself by default. This means, that if you wanted to share the document with someone else, you would also have to share the images.

The advantage is that the images will automatically be updated if the original is changed.

If you want to embed the images into the document, then you can use the `Edit:Links:Break Link` menu item (or something similar depending on your application) to embed each image as desired.

Image size

Images work best in OpenDocument when both a height and width attribute are specified. If only a width or height is specified, then things will work, but the image might not look

correct. If you specify a dimension as a percentage, you may have to manually adjust the image to fit the guidelines provided in the document.

6.4 OPML

OPML, or Outline Processor Markup Language, is a basic format that is useful for importing and exporting from various outlining programs (e.g OmniOutliner[1]). When used with MultiMarkdown, you can switch back and forth from an outliner and text editor as necessary to work on a document. The text editor will likely be easier for writing your prose, and the outliner may be easier when rearranging the structure of your document.

"Irregular" hierarchy issues

When using an Outliner, each level in the document (e.g. `h1`, `h2`, etc.) is a child of a parent that is one level "higher" than the child. For example:

```
## Second Level ##

### Third level ###

This is a child of the level 2 parent, "Second Level"
```

It causes problems if you were to try and "skip" a level:

```
## Second Level ##

#### Fourth level ####

This is *not* a child of the level 2 parent, "Second Level"
```

If you use `multimarkdown` to process a text file containing such "skips" into an OPML, any "orphaned" levels will not be included in the output. This can be seen as a feature, or a limitation depending on your needs.

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

Chapter 7

What's different in MultiMarkdown 3.0?

7.1 Why create another version of MultiMarkdown?

- Maintaining a growing collection of nested regular expressions was going to become increasingly difficult. I don't plan on adding much (if any) in the way of new syntax features, but it was a mess.
- Performance on longer documents was poor. The nested perl regular expressions was slow, even on a relatively fast computer. Performance on something like an iPhone would probably have been miserable.
- The reliance on Perl made installation fairly complex on Windows. That didn't bother me too much, but it is a factor.
- Perl can't be run on an iPhone/iPad, and I would like to be able to have MultiMarkdown on an iOS device, and not just regular Markdown (which exists in C versions).
- I was interested in learning about PEG's and revisiting C programming.
- The syntax has been fairly stable, and it would be nice to be able to formalize it a bit — which happens by definition when using a PEG.
- I wanted to revisit the syntax and features and clean things up a bit.
- Did I mention how much faster this is? And that it could (eventually) run on an iPhone?

7.2 "Complete" documents vs. "snippets"

A "snippet" is a section of HTML (or LaTeX) that is not a complete, fully-formed document. It doesn't contain the header information to make it a valid XML document. It can't be compiled with LaTeX into a PDF without further commands.

For example:

```
# This is a header #

And a paragraph.
```

becomes the following HTML snippet:

```
<h1 id="thisisaheader">This is a header</h1>

<p>And a paragraph.</p>
```

and the following LaTeX snippet:

```
\part{This is a header}
\label{thisisaheader}


And a paragraph.
```

It was not possible to create a LaTeX snippet with the original MultiMarkdown, because it relied on having a complete XHTML document that was then converted to LaTeX via an XSLT document (requiring a whole separate program). This was powerful, but complicated.

Now, I have come full-circle. peg-multimarkdown will now output LaTeX directly, without requiring XSLT. This allows the creation of LaTeX snippets, or complete documents, as necessary.

To create a complete document, simply include metadata. You can include a title, author, date, or whatever you like. If you don't want to include any real metadata, including "format: complete" will still trigger a complete document, just like it used to.

NOTE: If the *only* metadata present is `Base Header Level` then a complete document will not be triggered. This can be useful when combining various documents together.

The old approach (even though it was hidden from most users) was a bit of a kludge, and this should be more elegant, and more flexible.

7.3 Metadata Differences

When metadata was repeated in MultiMarkdown 2.0, the second instance "overwrote" the first instance. In MultiMarkdown 3.0, each instance of a repeated metadata key is used. This is necessary for something like `LaTeX Input` which can require multiple instances.

This means you can't "erase" an unnecessary metadata value by including a second, empty, copy. This was a trick used, for example, to erase undesired style information inserted into the document by older versions of Scrivener.

7.4 Creating LaTeX Documents

LaTeX documents are created a bit differently than under the old system. You no longer have to use an XSLT file to convert from XHTML to LaTeX. You can go straight from MultiMarkdown to LaTeX, which is faster and more flexible.

To create a complete LaTeX document, you can process your file as a snippet, and then place it in a LaTeX template that you already have. Alternatively, you can use metadata to trigger the creation of a complete document. You can use the `LaTeX Input` metadata to insert a `\input{file}` command. You can then store various template files in your texmf directory and call them with metadata, or with embedded raw LaTeX commands in your document. For example:

```
LaTeX Input:        mmd-memoir-header
Title:              Sample MultiMarkdown Document
Author:             Fletcher T. Penney
LaTeX Mode:         memoir
LaTeX Input:        mmd-memoir-begin-doc
LaTeX Footer:       mmd-memoir-footer
```

This would include several template files in the order that you see. The `LaTeX Footer` metadata inserts a template at the end of your document. Note that the order and placement of the `LaTeX Include` statements is important.

The `LaTeX Mode` metadata allows you to specify that MultiMarkdown should use the `memoir` or `beamer` output format. This places subtle differences in the output document for compatibility with those respective classes.

This system isn't quite as powerful as the XSLT approach, since it doesn't alter the actual MultiMarkdown to LaTeX conversion process. But it is probably much more familiar to LaTeX users who are accustomed to using `\input{}` commands and doesn't require knowledge of XSLT programming.

I recommend checking out the default LaTeX Support Files[1] that are available on github. They are designed to serve as a starting point for your own needs.

Note: You can still use this version of MultiMarkdown to convert text into XHTML, and then process the XHTML using XSLT to create a LaTeX document, just like you used to in MMD 2.0.

7.5 Images

In HTML, images have three pieces of "metadata" relevant to MMD — `alt`, `title`, and `id`. In Markdown and MultiMarkdown 2.0, these were structured in the following manner:

```
This is an image ![alt text](file.png "This is a title")
```

In MultiMarkdown 2.0, the "alt text" was processed to "alttext" and used as an `id` attribute.

The problem was that LaTeX, and later on ODF, didn't really need the `alt` or `title` metadata — those formats really needed a *caption* instead. Using a caption in some formats, but not others, leads to strange inconsistencies in the documents created from the same MultiMarkdown source. And while I understand the differences between the `alt` and `title` attributes, it really doesn't make a lot of sense.

So instead, in MultiMarkdown 3.0, the following is used instead:

```
This is an image ![This is alt text](file.png "This is a title")

or

This is an image ![Another *alt*][fig]

![This is a *caption*][fig2]
```

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

```
The following image has no ‘alt‘ or ‘caption‘:

![][fig2]

[fig]: file2.png "This is another title"
[fig2]: file3.png "This is another title"
```

The `id` attribute comes into play when an image is specified as a reference. This allows you to link to the image from elsewhere in your document.

When an image is the only thing constituting a paragraph, it is becomes wrapped in a <`figure`> tag, and instead of an `alt` attribute, it has a caption. This is demonstrated by `fig2` above. This caption is used regardless of output format, providing consistency between HTML, LaTeX, and ODF. When this happens, the alt tag is a stripped down version of the caption, since it can't have any markup applied.

So, in MultiMarkdown 3.0, there are now 4 pieces of metadata — `alt`, `title`, `id`, and an optional `caption`. There are only three places to describe this metadata, so one piece has to be duplicated. Currently, the `alt` is a duplicate of the `caption`, sans any markup. If an image is contained within a paragraph, an `alt` attribute is created, but no `caption`.

An alternative plan I am considering is to use what currently generates the `title` attribute to instead generate the `alt` attribute in all instances, and the caption can be ignored if an image is not considered a figure.

7.6 Footnotes

Footnotes work slightly differently than before. This is partially on purpose, and partly out of necessity. Specifically:

- Footnotes are anchored based on number, rather than the label used in the MMD source. This won't show a visible difference to the reader, but the XHTML source will be different.
- Footnotes can be used more than once. Each reference will link to the same numbered note, but the "return" link will only link to the first instance.
- Footnote "return" links are a separate paragraph after the footnote. This is due to the way peg-markdown works, and it's not worth the effort to me to change it. You can always use CSS to change the appearance however you like.
- Footnote numbers are surrounded by "[]" in the text.

7.7 Raw HTML

Because the original MultiMarkdown processed the text document into XHTML first, and then processed the entire XHTML document into LaTeX, it couldn't tell the difference between raw HTML and HTML that was created from plaintext. This version, however, uses the original plain text to create the LaTeX document. This means that any raw HTML inside your MultiMarkdown document is **not** converted into LaTeX.

The benefit of this is that you can embed one piece of the document in two formats — one for XHTML, and one for LaTeX:

```
<blockquote>
<p>Release early, release often!</p>
<blockquote><p>Linus Torvalds</p></blockquote>
</blockquote>

<!-- \epigraph{Release early, release often!}{Linus Torvalds} -->
```

In this section, when the document is converted into XHTML, the `blockquote` sections will be used as expected, and the `epigraph` will be ignored since it is inside a comment. Conversely, when processed into LaTeX, the raw HTML will be ignored, and the comment will be processed as raw LaTeX.

You shouldn't need to use this feature, but if you want to specify exactly how a certain part of your document is processed into LaTeX, it's a neat trick.

7.8 Math Support

MultiMarkdown 2.0 supported ASCIIMathML[2] embedded with MultiMarkdown documents. This syntax was then converted to MathML for XHTML output, and then further processed into LaTeX when creating LaTeX output. The benefit of this was that the ASCIIMathML syntax was pretty straightforward. The downside was that only a handful of browsers actually support MathML, so most of the time it was only useful for LaTeX. Many MMD users who are interested in LaTeX output already knew LaTeX, so they sometimes preferred native math syntax, which led to several hacks.

MultiMarkdown 3.0 does not have built in support for ASCIIMathML. In fact, I would probably have to write a parser from scratch to do anything useful with it, which I have little desire to do. So I came up with a compromise.

ASCIIMathML is no longer supported by MultiMarkdown. Instead, you *can* use LaTeX to code for math within your document. When creating a LaTeX document, the source is simply passed through, and LaTeX handles it as usual. *If* you desire, you can add a line to your header when creating XHTML documents that will allow MathJax[3] to appropriately display your math.

Normally, MathJax *and* LaTeX supported using `\[ math \]` or `\( math \)` to indicate that math was included. MMD stumbled on this due to some issues with escaping, so instead we use `\\[ math \\]` and `\\( math \\)`. See an example:

```
latex input:    mmd-article-header
Title:          MultiMarkdown Math Example
latex input:    mmd-article-begin-doc
latex footer:   mmd-memoir-footer
xhtml header:   <script type="text/javascript"
    src="http://localhost/~fletcher/math/mathjax/MathJax.js">
    </script>


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

[2] http://www1.chapman.edu/~jipsen/mathml/asciimath.html
[3] http://www.mathjax.org/

```
And an equation on it's own:

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

That's it.
```

You would, of course, need to change the `xhtml header` metadata to point to your own installation of MathJax.

Note: MultiMarkdown doesn't actually *do* anything with the code inside the brackets. It simply strips away the extra backslash and passes the LaTeX source unchanged, where it is handled by MathJax *if* it's properly installed, or by LaTeX. If you're having trouble, you can certainly email the MultiMarkdown Discussion List[4], but I do not provide support for LaTeX code.

[4] http://groups.google.com/group/multimarkdown/

Chapter 8

Known Issues

8.1 Non-ASCII characters require complete HTML document

In order for Non-ASCII characters to display properly in most browsers, there needs to be some indication of the character set being used. Without getting into too many technical details, the easiest way to get HTML output to display properly is to include metadata that will trigger a complete document, e.g. a `Title`.

8.2 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.

8.3 Math Support lacking in OpenDocument

I figured out how to embed MathML in an OpenDocument file. There are several tools to convert LaTeX to MathML, including blahtex.

I'll probably start by writing a perl utility that can take a MMD-created ODF, and then parse the math "bits" out and put them back in as MathML.

Even better would be if I could figure out how to compile blahtex into multimarkdown as a library to do the processing internally.... Not sure how long that will take to get around to. Anyone with any particular skills in that area who wants to help - let me know!

8.4 Bibliography support lacking in OpenDocument

There is no automatic processing of citations when creating an OpenDocument file. Suggestions are welcome, as I would like to come up with a reasonable solution, if possible.

8.5 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.

Chapter 9

FAQ

Answers to common questions about MultiMarkdown. Contributions welcome!

NOTE: This section of the documentation is still quite a mess. Please feel free to contribute to fixing it up on the wiki[1].

9.1 Mac OS X FAQ

MultiMarkdown won't work with TextMate

If you get this error:

```
/bin/bash: line 2: multimarkdown: command not found
```

then you need to set the path properly in your preferences to include `/usr/local/bin` as shown in the screenshot (Figure 9.1).

9.2 Windows FAQ

How do I create a Drag and Drop app for Windows?

Under Windows XP, go to the location where you would like to place the shortcut, and then:

- right-click
- select "New->Shortcut"
- enter "multimarkdown" as the location of the item
- choose the desired name for the shortcut
- right-click on the new shortcut and select "Properties"
- you can then add any desired command-line options at the end of the "Target" property — you should definitely include "-b" to enable batch mode. The others are up to you.

[1] https://github.com/fletcher/peg-multimarkdown/wiki/User's-Manual

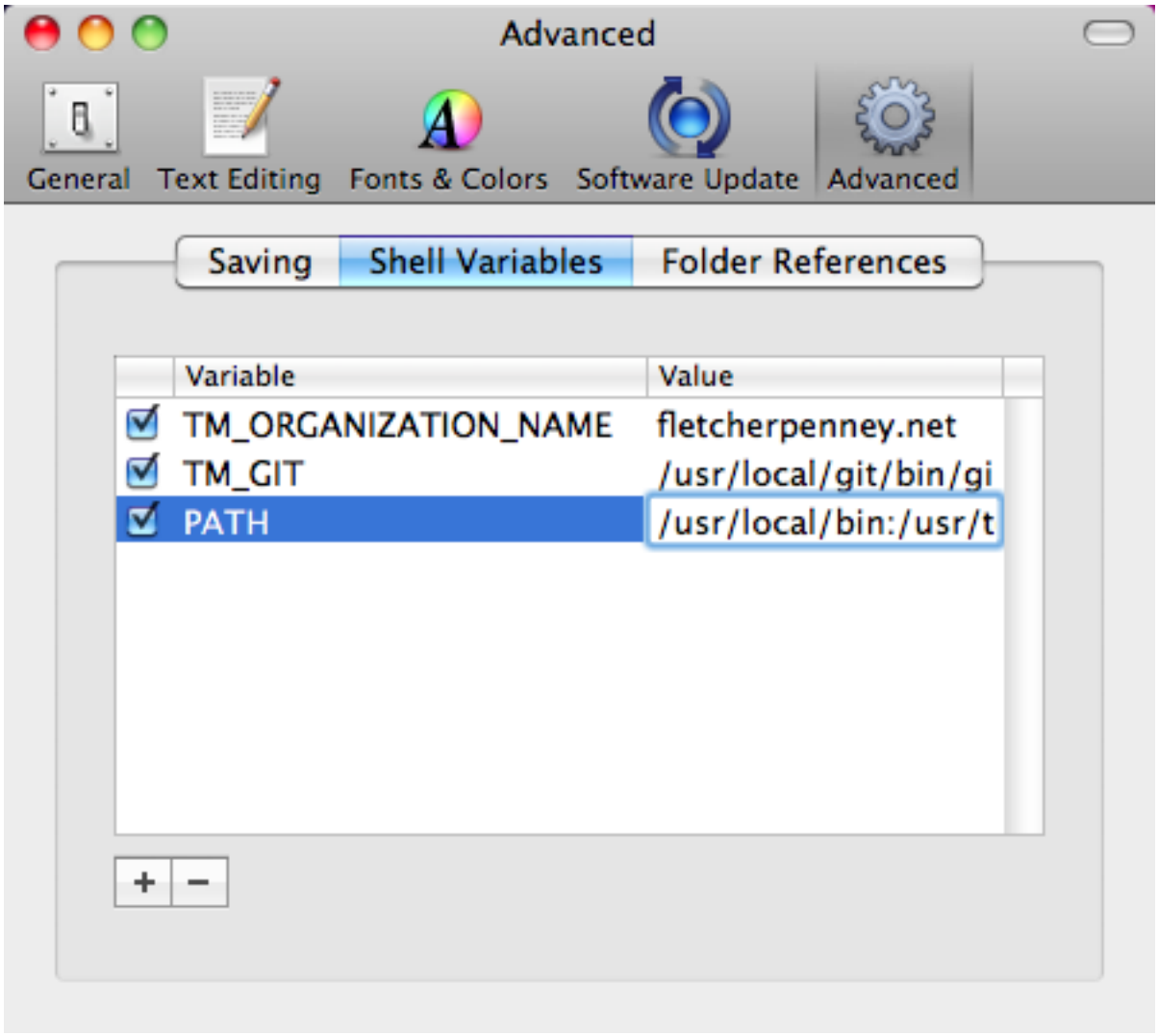

Figure 9.1: This is a screenshot of the relevant preferences

9.3 Linux FAQ

9.4 HTML FAQ

When I try to convert to HTML, my document ends up in the metadata

You are using an automated tool to create "complete" content. The tool is adding the `format: complete` metadata for you, but you don't have any other metadata in your document.

Start your document with a blank line to indicate that there is no more metadata.

This should only happen with tools designed for MultiMarkdown 2.0

Why don't non-ASCII characters display properly?

In order to display non-ASCII characters, a "complete" HTML document is necessary — a "snippet" doesn't work. Simply include some metadata to force complete document generation.

This is true for the QuickLook Generators as well.

9.5 LaTeX FAQ

What is LaTeX and how do I use it?

LaTeX[2] is a syntax for describing how to convert marked up plain text into high-quality, typeset documents. MultiMarkdown uses LaTeX to generate PDF's from your MMD document.

Of note, MultiMarkdown does not process LaTeX files for you. It generates them, and relies on you to convert the LaTeX into a PDF. The exact commands you need may vary, but I use the following sequence in my helper script to include all the features I use.

```
latexmk file.tex            # This command will do most of the work
makeglossaries file.tex     # process glossary entries if they exist
pdflatex file.tex           # I found that sometimes an extra run
pdflatex file.tex           # or two is needed
latexmk -c file.tex         # clean up a bunch of the temp files
```

How can I use raw LaTeX (or other code) with MultiMarkdown?

Anything placed in an HTML comment will be passed along unaltered - this can include raw LaTeX code. For example:

```
This is raw LaTeX: <!-- e^{i \cdot \pi} = -1 -->
```

will generate:

```
This is raw LaTeX: e^{i \cdot \pi} = -1
```

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

See the section on MultiMarkdown and LaTeX (section 3.4) for more information.

My LaTeX won't compile — what's wrong?

First - the vast majority of the time, this is not really a MMD problem, but rather a LaTeX problem, or (honestly) user error. Very occasionally, there are instances where MMD produces invalid LaTeX output when given unusual input, but these are rare.

Check to make sure you have proper metadata configured to allow your LaTeX to compile. See sample documents from the MultiMarkdown Gallery[3] for some examples of what metadata is required for various document types/

If you are having problems with a more "complicated" MMD tool (such as the TextMate bundle, or the Drag and Drop versions), use the command line approach when you are really having difficulties. See if your document will compile to HTML using the `multimarkdown` command. Once you've got that working, try moving on to LaTeX, and then a PDF (if that's your goal).

If that doesn't work, you need to track down the source of the error. The best approach is similar to the old program Conflict Catcher. Keep dividing your document into halves, and test each half. One half will likely compile without complaint, and the other won't. Take the problem half, and repeat this process until you are able to find the paragraph, sentence, word that causes the error.

Using a real text editor (e.g. TextMate[4]), examine the trouble section and make sure to view the invisible characters, and look for something odd. If you can't do that, you can also delete the faulty sentence and retype it (*don't* copy and paste — you'll simply replicate the problem).

If none of that fixes your problem, post the problem section to the MultiMarkdown Discussion List[5] and you will likely find out what you're doing wrong. Or perhaps you found an error and I will fix it. ;)

TextMate won't produce a PDF

MultiMarkdown 2.0 included a command to create a PDF "automagically" from a MultiMarkdown file. This led to a lot of confusion and questions from users who didn't understand what was going (e.g. that MMD creates a LaTeX file, and that pdflatex then creates the PDF).

MultiMarkdown 3.0 does not *officially* support the creation of a PDF. It supports the creation of a *LaTeX* file. You are then responsible for creating the PDF.

That said, there is a script `mmd2pdf` that runs some of those commands for you. If you have trouble with it, you're on your own. Try running the appropriate LaTeX commands in the Terminal by hand to create your PDF.

[3] https://github.com/fletcher/MultiMarkdown-Gallery
[4] http://macromates.com/
[5] http://groups.google.com/group/multimarkdown

9.6 OPML FAQ

9.7 OpenDocument FAQ

9.8 Syntax FAQ

How do I split a MultiMarkdown document into several parts?

MultiMarkdown is designed to process a single document at a time, but sometimes it can be useful to split a longer document into multiple parts. For example, this User's Guide is split into multiple pages on a wiki, and then combined for processing with MultiMarkdown.

In the MMD-Support[6] package, there is a utility script `mmd_merge` that can stitch multiple text files together. It reads an input file that describes which files need to be combined, in what order, and at what "level". This allows you to ensure that header levels are adjusted appropriately when the document is combined.

For example, the `index.txt` file used to combine this User's Guide is:

```
# User's guide #

metadata.txt

Introduction.md

How-do-I-install-MultiMarkdown?.md

How-do-I-use-MultiMarkdown?.md

How-do-I-create-a-MultiMarkdown-document?.md

How-do-I-customize-MultiMarkdown-Output?.md

MultiMarkdown-Output-Formats.md

What's-different-in-MultiMarkdown-3.0?.md

Known-Issues.md

FAQ.md

Acknowledgments.md
```

As you see above, I use a separate text file for storing metadata. Lines that begin with `#` are considered comments.

If you want to indicate that a certain file should have it's header levels increased, you can insert one or more tab characters before the filename. This is analogous to changing the `Base Header Level` metadata for that file.

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

NOTE: `mmd_merge` is a Perl script, so it requires that you have Perl installed in order to use it.

How do I link to an image?

If you want to link to an image included in your document, you should define the image using the reference syntax:

```
This is an image ![Alt Text][image label].

[image label]: /path/to/image.png "This is a title"

And now a link the desired [image](#imagelabel).
```

Reference id's don't work with hyphens

Without getting too technical, you have to be careful when including multiple hyphens and specifying links or images by reference.

```
This is an ![image-1].

This is the same image, but looks different --- ![image--1].

This is a different image ![image---1].

[image-1]: file.png     "This is used"
[image--1]: file2.png   "This is ignored"
[image---1]: file3.png  "This is used"
```

This is because - and - are both converted to `en-dash`, whereas -- is an `em-dash`.

9.9 General FAQ

How do I install MMD?

Please see How do I install MultiMarkdown? (chapter 2).

How do I uninstall MMD?

The Mac installer for MultiMarkdown installs the following files:

- /usr/local/lib/libglib–2.0.0.dylib
- /usr/local/lib/libintl.8.0.2.dylib
- /usr/local/lib/libintl.8.dylib
- /usr/local/lib/libintl.dylib

- /usr/local/bin/multimarkdown
- /usr/local/bin/mmd
- /usr/local/bin/mmd2odf
- /usr/local/bin/mmd2opml
- /usr/local/bin/mmd2pdf
- /usr/local/bin/mmd2tex

The Windows installer includes an uninstaller.
For Linux, simply remove the files from wherever you put them.

How can I better learn the MultiMarkdown Syntax?

The syntax of MultiMarkdown can be a challenge for beginners especially when it comes to using Table and Link syntax. The challenge comes in understanding how it looks *after* the conversion.

For Mac users, you can download a free program NValt (http://brettterpstra.com/code/) that contains a preview mode. Thus, you can enter your markdown with a dynamic preview window to show how the formatting appears. I've found this useful until I was comfortable with the syntax.

Note also that MultiMarkdown can also be installed to MarsEdit (http://red-sweater.com/marsedit/) which also has a preview mode. This is even more effective for for bloggers as it will handle inline image upload to most popular blog software and display the interpreted output of your MMD syntax.

If you're interested in web based learning, you may want to try http://peg.gd/ which provides a web site for writing in Markdown syntax and display. You can also save the pages for forwarding to other people.

For the reverse requirement, you can visit http://heckyesmarkdown.com/ which will attempt to take a given HTML Page and convert it into Markdown, usually with good accuracy.

Can I use MultiMarkdown on an iPhone/iPad?

There are a number of programs available for use on the iPhone and iPad that support regular Markdown, but not MultiMarkdown (yet!).

Elements[7] is a iPhone /iPad app that also has a markdown preview mode. Second Gear has also made MarkdownMail[8] which is simpler, but also very usable for learning Markdown.

Textastic[9] is a syntax highlighting text editor for the iPad that includes a preview mode for Markdown. It will even preview MathJax equations if you include the proper metadata and a working MathJax installation!

Now that MultiMarkdown is available in C (as of version 3.0), I would like to work on modifying it to work with iOS.

[7] http://www.secondgearsoftware.com/elements/
[8] http://www.secondgearsoftware.com/markdownmail/
[9] http://www.textasticapp.com/

How do I troubleshoot an error?

First, check some basics:

- are you using MultiMarkdown 2.0, or 3.0?
- are you using the latest release of your version?
- if you are using an application (Scrivener, TextMate, etc) does MultiMarkdown work from the command line?
- if you're using LaTeX, did you install the proper support files?
- did you read the User's Manual? (Yes — I hate manuals too, but some aspects of MultiMarkdown can be rather complex, and that's where most questions come up)
- is this a known issue on github[10], or the discussion list[11]?

If you're still having a problem, you first have to figure out what sort of error you're having:

- The source document crashes multimarkdown — this is *very* rare and can be confirmed by trying to convert text to HTML. Use the "Divide and Conquer (section 9.9)" approach to find the problem portion of the document and send it to the discussion list.
- You successfully generate LaTeX, but can't compile it — see the section on troubleshooting LaTeX (section 9.5) above.
- You successfully generate a document, but it won't open properly (this generally happens with OpenDocument, but could be other formats) — use the "Divide and Conquer (section 9.9)" approach to find the problem portion, and send it to the discussion list.
- The output you get doesn't match what you were expecting — read the pertinent section of the manual, and compare your results against some of the sample documents. Make sure your syntax usage is correct. If you're still having trouble, send an email to the discussion list.

If you are sending an error report to the discussion list, please be sure to do the following:

- narrow down the problem to the smallest example that replicates the issue
- be sure to include your input, the output you received, and the output you expected to get
- if you're convinced this is an issue with MultiMarkdown, use the issue tracker on github[12], otherwise the discussion list[13].

[10] https://github.com/fletcher/peg-multimarkdown/issues
[11] http://groups.google.com/group/multimarkdown/
[12] https://github.com/fletcher/peg-multimarkdown/issues
[13] http://groups.google.com/group/multimarkdown/

How do I find the source of an error?

When you have a longer document that MultiMarkdown can't successfully process, it is helpful to narrow down the actual source of the problem. I recommend that you recursively split the document into halves until you reach the source of the issue. I also call this technique "Divide and Conquer".

Basically, divide your document into two halves and try to process each one separately. If one half works, and the other doesn't, the problem is likely in the half that won't compile. Continue recursively dividing the problem piece in half, until you narrow in on the actual problem.

Keep in mind that some parts of the document would potentially need to be included in both halves for proper testing — the metadata, and the definition of any images or links by reference, for example.

Obviously, if an error requires two pieces of the document to occur, and one piece is in the beginning and the other is at the end, this approach might fail to localize the problem. So, it's not perfect, but is quite useful.

How do I get more help?

The MultiMarkdown web site[14] describes various ways to get help.

How can I contribute to MultiMarkdown?

The MultiMarkdown web site[15] describes various ways you support or contribute to the MultiMarkdown project.

Thanks for thinking about it!!

What ideas are there for the future of MultiMarkdown?

More complex syntax for tables

Several users have written with suggestions for additional syntax for tables in order to allow cells to span multiple rows, to allow the MMD source for a single table row to span multiple text rows, etc.

My concern here is that the table should be easily understood in plain text, without too much markup. Ideally, there would be no markup, but it's just too complicated to have MMD figure out what the layout means.

I welcome suggestions, but I plan to be somewhat strict on this one.... And remember, for complex tables you can always enter the raw HTML.

A syntax for adding forms?

This is less likely to happen, but it seems that adding some sort of basic syntax to manage forms would be pretty easy, and not very intrusive. Something like:

```
Please check all that apply:

[ ] Option A
```

[14] http://fletcherpenney.net/multimarkdown/help/
[15] http://fletcherpenney.net/support/

```
[ ] Option B
[ ] Option C

Please choose one of the following:

( ) Option A
( ) Option B
( ) Option C

Please enter your first name:

[_____________________]
```

The idea would be to allow you to easily enter the necessary fields for a form, but not have to be concerned with the HTML nonsense.... As always, input welcome.

How do I submit an idea for MultiMarkdown?

First, my goal with MultiMarkdown is *not* to continue to add scores of features. More is not always better.

If there is a feature that you can't live without, and you are interested in submitting your idea, think through the following:

- Can your idea be done with the existing setup — for example, using XSLT or a quick post-processing script to convert the default output to the output you desire?
- Does the raw source you are proposing look natural to a human reader? The less markup the better — the raw MultiMarkdown should look like something written by and for people, not for a computer.
- How many people need this feature? MultiMarkdown has now been around for quite a while — why hasn't someone else already requested this feature if it's so important?

If you feel your idea should be included in MultiMarkdown, then send it to the discussion list and I will certainly review it.

What is a PEG?

A PEG, or Parsing Expression Grammar[16] is a method of describing a formal grammar to parse an input language. John MacFarlane used a PEG to create his peg-markdown[17]. I then modified his work to create MultiMarkdown 3.0.

What happened to RTF output?

First — RTF is really an inferior document format. A given RTF document will give very different appearances when opened with different applications, or even in the same application on different operating systems. Apple's implementation of RTF is very different from Microsoft's, which is different from that of everyone else. It's hard to take RTF

[16] http://en.wikipedia.org/wiki/Parsing_expression_grammar
[17] https://github.com/jgm/peg-markdown

seriously as a document format when it seems to be handled differently everywhere you look.

Ironically, MultiMarkdown never *really* had RTF support to begin with. The initial approach to generating RTF output was to convert to HTML, and then use Apple's `textutil` command to convert to RTF. This can still be done via the terminal, or simply use TextEdit to open the HTML and then save as RTF — this effectively does the same thing, except that this approach can also handle images.

A second approach was started, but never completed — the `xhtml2rtf.xslt` file was used to convert HTML to RTF via an XSL stylesheet. This approach was never finished, but could also be updated to work with MMD 3 generated HTML output.

Because of all this, RTF "support" is no longer included with MMD. You can easily modify a shell script to pipe the HTML command through the `textutil -convert rtf -stdout` command and save to the desired filename, just like in MMD 2.0. But an approach that gives better results is to use the Flat OpenDocument format instead. This can be opened in LibreOffice or OpenOffice, and then saved to a large variety of document formats.

A better approach might be to stick with the `.fodt` format, or the more common version — `.odt`. But that's more of a "political" opinion than a technical one.

Chapter 10

Acknowledgments

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[1] and helping me get started on MMD 3.0!)

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

- David Sparks
- Katie Floyd
- Daniel Müller

and others I have surely forgotten. . . .

In addition to the core `peg-markdown` code, MultiMarkdown 3.0 makes use of the `glib2` and `intl` libraries that are a part of GTK+[2] and licensed under the GNU LGPL 2.1. Also, it relies on `peg` and `leg`[3] to create the actual parser.

[2] http://www.gtk.org/
[3] http://piumarta.com/software/peg/