A toad's guide to using Scrivener – MultiMarkDown – Latex

Huw Evans

January 26, 2012

Formatted for LaTeX
by MultiMarkdown

Contents

Chapter 1

To begin

This document is intended as an introductory guide to producing PDF documents using the Scrivener 2.2 for Mac, MultiMarkdown 3.x[1] and Latex workflow (hereafter ScML). All three of those programs have manuals and guides, this document describes how to get them to work together, so you can get on with your writing.

The guide comes out of my experience over the last few years, and the frustrations – and joys – of using the ScML workflow to produce good-looking PDFs. Some of those frustrations were my own fault (a few too embarrassing to even discuss), but others stem from the fact that getting the ScML workflow right means getting three separate systems to line up at program level and syntax level. As a non-programmer it felt at times as if I was setting up a Honda advert[2].

I hope this guide will help a few people get past those initial hiccups so they can experience the benefits of the ScML workflow for themselves.

But where does the toad come into it? There's a proverb, put this way by Rudyard Kipling:

> The toad beneath the harrow knows
>
> Exactly where each tooth-point goes;

I am writing as one toad (user) to other toads to help them get past some of the spikes (problems).

1.1 Acknowledgements

But before we plunge in, there are a few *without whoms*:

- Keith (not Kevin) Blount the creator and coder of Scrivener, the best writing tool this planet has yet seen.
- Fletcher Penney the creator of MultiMarkdown.
- The contributors to the Scrivener MultiMarkdown forum whose knowledge and advice have helped me avoid several spikes.

1.2 Why would you?

Latex turns out lovely-looking documents. But writing an input tex file by hand is a real pain. It's the sort of experience that drives people to use word processors where they can click one button to turn a word bold. (And look, by jingo, it even goes bold on screen!)

[1] There was a big change in MultiMarkdown between versions 2 and 3. The instructions in the rest of this document won't work with version 2.

[2] http://www.youtube.com/watch?v=_ve4M4UsJQo

Scrivener is designed to let writers write without worrying about fonts, layouts and all the other stuff until the writing is done.

Wouldn't it be great if there was a way of getting your writing done in Scrivener then being able to use all the text processing power of Latex without having to shovel lots of code?

That's where MultiMarkdown comes in. You could think of it as glue which holds Scrivener and Latex together, or – more helpfully – as a wonderful piece of plumbing which lets your words flow from Scrivener to Latex. (So when you open your brain tap … No, that's taking a metaphor too far.)

The Scrivener – MultiMarkDown – Latex workflow (plumbing again) minimises distractions while you write and lets you get well-laid out documents at the end. It takes a bit of setting up, as there are three programs to install and configure properly, but the end result makes it worth the effort.

The rest of this guide takes you briefly through the installation of the software, then looks at the three stages of:

- writing;
- compiling;
- typesetting

But first, we need to grasp the ScML workflow as a whole.

1.3 An overview of the ScML workflow.

Before we plunge into the detail we need to take an overview of the workflow. But before we do that we need to make one other thing clear:

> MultiMarkdown refers to both a set of markup syntax *and* a program which processes text containing that syntax.

I'll try to make clear which MultiMarkdown I'm referring to at any one points.

So, the workflow goes:

1. Write text in Scrivener, using most of its wonders (but **not** the automatic section numbering).
2. Markup the text with MultiMarkdown syntax and, if you need it, embedded Latex syntax.
3. Compile the text in Scrivener and process through MultiMarkdown to get a tex file.
4. Open the tex file in TexShop and typeset to create a PDF.

Of course, there'll be something wrong somewhere so you'll go round the loop a couple of times, but at the end you'll still get that PDF. (You can also send the working tex file to a journal if you're an academic).

Chapter 2

Setting things up

To get ScML working you need the three programs installed on your Mac.

2.1 Scrivener

You've probably already installed Scrivener – if you haven't, download it[1] and give it a go: there's a very generous thirty (non-consecutive) day trial period.

2.2 MultiMarkdown

When it comes to setting up MultiMarkdown there's the easy way or the hard way. If you want to go the hard way, fine, the MultiMarkdown manual is there for you. But I'm going the easy way: download the latest version of the MultiMarkdown-Mac installer[2] and run the installer.

Because we're running MultiMarkdown through Scrivener there are a couple of extra things we need to install:

1. Download and run the MultiMarkdown-Support-Mac installer[3]
2. Download the LaTeX support files[4] and place them in ~/Library/texmf/tex/latex/mmd.

The Latex support files are tex files which MultiMarkdown uses to create complete tex files during typesetting. We'll cover the use of those support files in more detail in section 4.1 and section 5.3.

2.3 Latex

There are approximately five hundred million ways of installing Latex[5] on a Mac. I only know one: MacTex[6]. That gives you a Tex engine for typesetting and the TexShop front end for fiddling on.

Once you have the basic Latex installation you will inevitably start to customise it by adding new *packages* which allow you to make the output even more lovely. We will talk more about Latex packages in section 5.4.

[1] http://www.literatureandlatte.com
[2] http://github.com/fletcher/peg-multimarkdown/downloads
[3] http://github.com/fletcher/peg-multimarkdown/downloads
[4] https://github.com/fletcher/peg-multimarkdown-latex-support
[5] I really don't want to get tangled in a terminology thing, so I'll refer to Latex as the code and Tex and TexShop as the software. The files we work with have the .tex extension so we'll call those tex files.
[6] http://www.tug.org/mactex/2011/

Chapter 3

Writing (Scrivener)

You already know about writing, so this section isn't written to nag you about split infinitives and the subjunctive, instead it's designed to help you maximise the time you spend in Scrivener and minimise the time spent wrestling with Latex.

3.1 The small stuff

This section covers the paragraph level syntax for MultiMarkdown.

Paragraphs

Latex recognises a new paragraph by a blank line with two returns: it seems to think one return is just dithering. So when you are writing in Scrivener make sure you give that extra press of the return key.

Emphasis

Or – bold and italics. Which is how we non-semantic coding people think of it.

To turn text **bold** just put two asterisks before and after the text, like this `**bold**`. Be careful not to leave a space between the asterisks and the first or last letter of the text, otherwise you'll end up with asterisks liberally scattered around your words.

To turn text *italic* just put one asterisk before and after the text, like this `*italic*`. The same warning on spacing applies.

Bullets and lists

Easy. For bulleted lists start the each paragraph with a + and a space. That's it. Dashes will do the job just as well:

```
- first bullet

- second bullet

- third bullet
```

gives

- first bullet
- second bullet

- third bullet

For numbered lists, start each paragraph with `1.` You can use other numbers if you want, but there's no need to worry about keeping the numbers in order. ScML will do that for you:

```
1. First important point

1. Second important point

1. Third important point
```

gives:

1. First important point
2. First important point
3. First important point

We can also have sub-indents by tabbing, like this:

```
1. First point

    + sub-bullet

    + another sub-bullet

1. Second point
```

Which becomes:

1. First point
    - sub-bullet
    - another sub-bullet
2. Second point

We can also have other paragraphs at the same indent as a bullet or list, by starting the paragraph with a tab. Like this:

```
+ first bullet

    linked paragraph

+ second bullet
```

Which gives:

- first bullet
  linked paragraph
- second bullet

Quotes `n' stuff

To get a block quote start the paragraph with a >:

```
>Someone else's interesting thought
```

To get something like:

> A man ceases to be a beginner in any given science and becomes a master in that science when he has learned that *this expected reversal is never going to happen* and that he is going to be a beginner all his life.
>
> R. G. Collingwood. *The New Leviathan*: 1.46.

Code blocks start with a tab. Small pieces of code can be shown with backquotes `.
MultiMarkdown special characters can be escaped with a backslash:

```
This is how I get the backquote \` to show up on the page.
```

3.2 Document structure

Most long documents have a formal structure of chapters or sections with sub-chapters, sub-sections, sub-sub-sections and so on. In technical documents those divisions will usually be numbered. Using numbered divisions:

- helps the reader understand the flow of the document;
- makes cross-referencing easier by linking to structure not pages.

But developing and maintaining structure and numbered divisions is a pain, particularly as a document changes structure during the writing process[1].

ScML overcomes this by using the structure of the Binder in the Scrivener project as the structure of your document. So, in this project, the Binder document titled `Writing in Scrivener' becomes a top level division, `The small stuff', which is nested in it becomes a second level section and `Emphasis' which is nested in that becomes a third level section. Figure 3.1 makes that point visually.

And that's it. All you have to do is make sure the structure in the Binder is right.

Now comes the magic of ScML: the titles of those documents become the numbered divisions in the final document. The pain of remembering if a section should be 5.1.2 or 5.2.2 disappears, as does the pain of renumbering everything if you put a new section in[2].

Headings from scratch

Just so you know, if you were to be writing a MultiMarkdown document from scratch, the document structure is built up using headers, which are marked by hashes to front and back:

```
#Heading#
```

One hash gives a top-level heading, two hashes a second level heading, and so on. If you look in the Scrivener project you'll see that the heading of this particular section isn't a separate document in the binder.

[1] If it doesn't then you're doing it wrong, or you are working to someone else's restrictive structure (most journals).

[2] If you look carefully at Figure 3.1 you'll see that the order of documents in the image doesn't match the order of this final document. What did I have to do to update the numbering once I'd changed the order? Nothing – I just compiled it again.

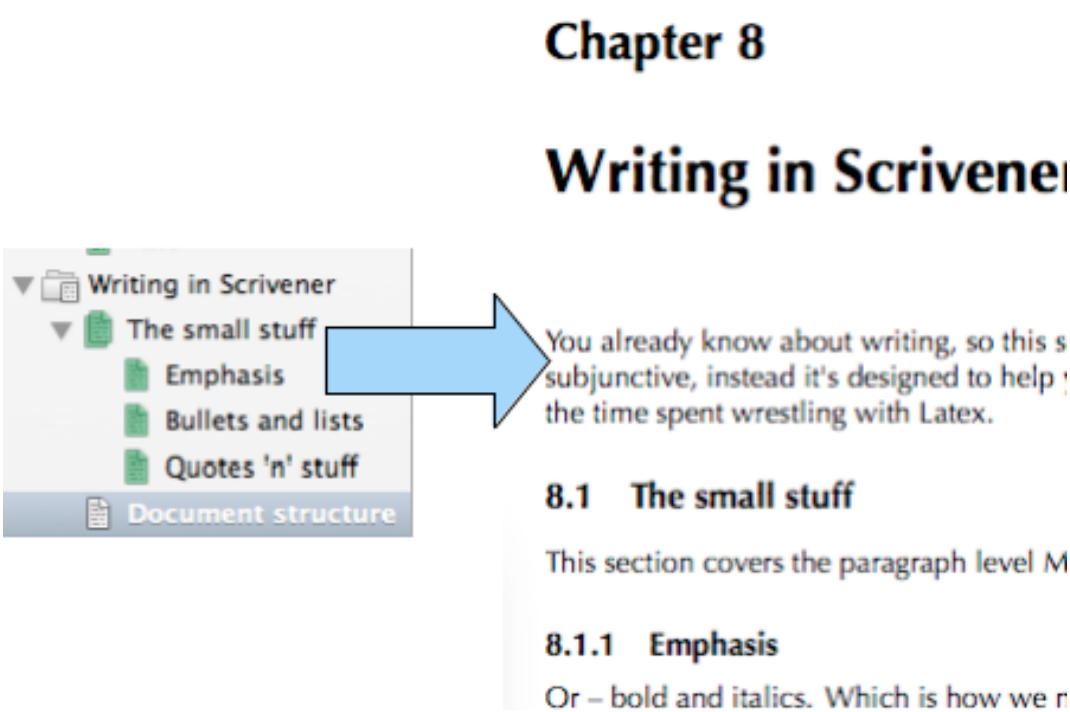

Figure 3.1: From the Binder to the Numbered sections

3.3 Cross-references

`That's all very well, but if the division numbers don't appear until the final PDF how can I make cross references?'

Well now, this is where MMD comes into its own.

The basic syntax of a cross-reference to another section of a document is, according the the MMD manual, two sets of square brackets with the title of the linked-to section inside the first set of brackets, like this:

```
[Document Title][]
```

So:

```
[Emphasis][] describes how to get italics.
```

Gives:

Emphasis (section 3.1) describes how to get italics.

While:

```
[Bullets and lists][] describes lists.
```

Gives:

Bullets and lists (section 3.1) describes lists.

In those examples you'll see the reference number comes out to the same section (3.1 when I last ran this): that's because, by default, Latex is only numbering chapters and sections. If I adjusted the numbering level to go down to subsections then those two examples would indicate different subsections.

Another issue with that basic link syntax is that it shows the name, division type and division number in the text. Usually, I don't want that, I just want the division number in my text so it reads `have a look at x.x to understand that'. To get round that I always use:

```
[](#documenttitle)
```

(that's a pair of square brackets, opening bracket, hash sign – Alt+3 on my UK keyboard – the Scrivener document title in lower case with spaces stripped out, and a closing bracket). So:

```
When compiling (see [](#compiling)), be very alert.
```

Will produce:

When compiling (see chapter 4) be very alert.

The same basic syntax works for links to figures and tables.

If you have several sections with identical titles, which can happen if every chapter needs an overview and a summary, how can you make sure you are referring to the right one?

There's a trick for that. In the Binder you can add a unique identification tag after the proper title:

```
Contents [Contents Overview]
```

You can then cross-refer to `[](#contentsoverview)`, as in:

```
That's all explained totally clearly in [](#contentsoverview).
```

To get:

That's all explained totally clearly in section 4.1.

3.4 Links

It shouldn't be a surprise to find the syntax for links is closely related to that for cross-references. We mark the link in the text with `[LinkName][]` and put the content in a separate paragraph:

```
[LinkName]: http://www.evilcorporation.com
```

So, if we wanted a link to the Scrivener web site we'd have:

```
Download Scrivener from [LandLSite][]

[LandLSite]: http://www.literatureandlatte.com
```

Which in real life gives:

Download Scrivener from LandLSite[3]

This basic format uses the `[LinkName]` in the text, which looks a bit odd. We can make it read better by shifting the label to the second set of brackets and adding a title in the first:

```
Don't forget to use the [Scrivener forums][ScrivForum]

[ScrivForum]: http://www.literatureandlatte.com/forum
```

Which looks like:

Don't forget to use the Scrivener forums[4]

The syntax allows attributes to be added to links as attribute=value or attribute=``multi word value'' pairs after the link address. However, that doesn't seem to do anything for ScML, so I wouldn't bother.

3.5 Footnotes

All good documents need footnotes – and technical documents more than most[5]. Footnotes in MultiMarkdown are similar to links, with an anchor in the text which looks like this: `[^MyFirstFootnote]`

And the footnote content, which is a separate paragraph starting with the anchor text plus a colon:

```
[^MyFirstFootnote]: I am so proud of this footnote.
```

Exactly where you put the footnote content in your Scrivener document is up to you. I have generally put the footnote text at the bottom of the Scrivener document which references it, so as not to interrupt the flow of the paragraphs on screen. However, I am now starting to put it immediately below the anchored paragraph: the ease with which Scrivener allows me to split documents means I often end up with the footnote text two documents away from its anchor. That doesn't stop it working, but it does mean I lose track of the footnote.

3.6 Tables

MultiMarkdown tables are simple: there are no pretty boxes, just the content of the table, set out in rows, with the cells separated by pipe characters: `|`. The only tricky bit is that we need to add a head-body division line using dashes between the pipes. We can also give the table a caption in square brackets underneath the last row:

```
|A header cell|The next header cell|
|-------------|--------------------|
|First cell   |Second cell         |
[My first table]
```

This becomes:

The pipes don't have to line up, but it can be easier to use tabs to line them up to make the table easier to read while you are putting it together[6].

We can use the title in our references, `[My first table][]` or `[](#myfirsttable)` to get:

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

[4] http://www.literatureandlatte.com/forum

[5] Because that's where the jokes go.

[6] The table code doesn't have to be laid out as tidily as the examples here, I've just done that to make it easier to read.

Table 3.1: My first table

A header cell	The next header cell
First cell second line	Second cell second line

see My first table (Table 3.1) or, my preferred option:

see Table 3.1

We can define the head-body division lines using a special row which has dashes between pipes, and colons to control the orientation of the cell content for each column. A colon at the left gives left alignment, at the right gives right alignment; two colons give centre alignment.

We can merge cells by sitting the pipes together, so this (which is borrowed directly from Fletcher Penney's MultiMarkdown guide):

```
[The caption of the table][TableLabel]
|             |Grouping                   ||
First Header  |Second Header  |Third Header  |
--------------|:------------:|------------:|
Content       |*Long Cell*                 ||
Content       |Cell           | Cell        |

New section   |More           | Data        |
And more      |And more                    ||
```

Gives this:

Table 3.2: The caption of the table

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

A few things to note there:

- the blank row in the table triggers a new horizontal rule.
- we can manage without the first pipes on all except the first row of the table.
- the caption can go above or below the table
- we can add a label alongside the caption, wrapped in its own set of square brackets. The label is then used in cross-referencing, allowing us to have longer and more descriptive titles.

In my view It's better to use the `[](#tablelabel)` cross-reference syntax which gives:

See Table 3.2.

Using the `[TableLabel][]` syntax gives strange results.

The main limits of the MultiMarkdown table is that it only manages cells with the content on a single line: if you want anything more complicated you will have to build it yourself.

If a table doesn't work at all check the Corrections tab of the Scrivener preferences and see if the substitution *replace double hyphens with em-dashes* is ticked. If it is ticked then the substitution results in em-dashes in the header-table division row, which MultiMarkdown then can't recognise.

3.7 Images or figures

There are two parts to including images into finished Latex pdfs:

- adding the references to include them;
- getting the image file to the right place for TexShop to find it during typesetting.

We handle the first with MultiMarkdown syntax, the second using Scrivener links.

MultiMarkdown uses a two-part syntax for images. First there is the anchor in the text:

```
![The image caption][TheImageLabel]
```

That has two parts, the caption and the label, each in square brackets, with and exclamation mark to start. Elsewhere, in a separate paragraph, we put the details of the image file, like this:

```
[TheImageLabel]: ImageFileName.png width=999px
```

The key parts are the label, followed by a colon, then the name of the file **with the extension** and finally an attribute, which in this example sets the size of the image in pixels[7].

So:

```
![A grumpy badger][Grumpy]

[Grumpy]: Grumpy.png width=200px
```

Brings the image into the text – you'll see it somewhere on the page (I can't tell you exactly where, because Latex will do the layout for me). Of course, if you want to see the actual link which is used you will have to look at the Scrivener file which contains the source text.

Once the image is set up, we can reference it in the usual way:

```
See [](#grumpy) for an example image.
```

which gives

> See Figure 3.2 for an example of an included image.

That deals with the first part of the business, making the image reference for Latex. For the second part, making sure the image file is in the right place, we use Scrivener's linking system:

1. Import the image into the Research folder. You can organise them in sub-folders if there are a lot of them. Note that Scrivener will hide the extension in the Binder list.

[7] There are variations on this format, and the MultiMarkdown manual also contains an in-line format which some people may prefer. Personally, the non-in-line format works better for me and keeps the text paragraphs uncluttered.

Figure 3.2: A grumpy badger

2. Once you've added the image text, select the file name (including the extension[8]) and right click.
3. Select *Scrivener link* and drill down until you get to pick the image you want. The file name will now be underlined to show a link, and coloured to your link colour (mine's currently orange).
4. When you compile the document Scrivener will export all linked files into a folder alongside the tex document, so they are all at hand for typesetting.

3.8 Maths

MultiMarkdown maths is Latex maths. The only difference is an extra \ in the beginning and ending codes for the mathematical expression.

In line maths expressions start with `\\(` and end with `\\)` like this:

```
We can calculate the resistance, \\(R_g\\), from the thickness.
```

Which becomes:

We can calculate the resistance, R_g, from the thickness.

Normal maths expressions (if that's the right term) starts with `\\[` and end with `\\]`. In my experience I've found that it only works if there is a space between the beginning and ending tags and the expressions:

```
\\[ h_r = Eh_{r0} \\]
```

which becomes:

$$h_r = Eh_{r0}$$

The only other thing worth saying about maths expressions is that if you get weary of untangling complicated expressions (and your idea of complicated may be much further down the line than mine) then MathType from Design Science is a great visual tool for building expressions, which can then be pasted as Latex code into Scrivener. You don't think I'd do this by hand?

[8]Back in the day, before MMD 3, there was no need to add the extension: and once Scrivener is updated to play nicely with MMD 3 I hope that will be the case again.

$$h_r = \frac{h_{r0}}{\frac{1}{\varepsilon_1} + \frac{1}{\varepsilon_2} - 2 + \frac{2}{\left(1+\sqrt{1+d^2/b^2}-d/b\right)}}$$

3.9 Bits of Latex

There will come a point where you need more Latex than ScML can give you. When you get to that point you can throw in as much Latex you want, provided you enclose it with HTML comments, `<!--` to open and `-->` to close. Typically I use that to enter SI units using the SIUnitx package[9]:

```
The area exceeds <!--\SI{1200}{mm^{2}/m}-->
```

Produces:

The area exceeds 1200 mm^2/m

You could also use it if there is a table you have massaged in Latex: then you could paste the raw Latex for the table into your Scrivener document and wrap it up with comments: MMD will pass it through undigested. (An alternative solution would be to save the table in a tex file of its own and use the Latex input command in Scrivener – commented out of sight – to link to the file. It would then get brought in as you typeset the final Latex. The only downside is that you would have to make sure you moved the file into the right folder every time you recompiled from Scrivener.)

3.10 Something's missing

ScML does support bibliographies using Bibtex. Unfortunately I know nothing about that. There is some information about bibliographies in the MultiMarkdown manual.

If you're reading this and you know how to integrate Bibtex into ScML workflow and your willing either to write some straightforward instructions or give me enough information so I can expand this section please get in touch.

3.11 Making life easier

Two tips for making the writing part easier:

- In my Scrivener projects I have a Project Notes file which contains my MultiMarkdown cheat sheet. It's got link syntax, table syntax, lots of SI units and a few bits of HTML for when I need that. As I usually work with the Inspector open it only takes me a few seconds to remind myself of a half-forgotten piece of syntax.
- MultiMarkdown Composer (see chapter 6) is an application produced by Fletcher Penney[10] the father of MultiMarkdown. It's a stand-alone editor specifically designed for use with MultiMarkdown. It knows all the tags and has a preview mode for seeing what the processed text will look like. Whilst you can write whole documents in it, my main use for it is to paste in troublesome bits of code and work on them until they make sense. I also write more complex tables in it, as MultiMarkdown Composer keeps the columns nicely aligned. When I'm done I simply paste the text back into Scrivener.

[9] Of course, I have to make sure that I've adjusted my mmd-latex files to declare the package, see section 5.4.
[10] http://fletcherpenney.net/

Chapter 4

Compiling (Scrivener and MultiMarkdown)

You've written your document and it's finished, or it's time to get a draft out to your supervisor, colleague or editor. In Scrivener terminology it's time to compile the document and, with the help of MultiMarkDown, turn it into a tex document which you can then use in TexShop.

4.1 Compile options

Start the compile process by selecting **File**>**Compile**. When the **Compile** dialogue opens go straight to the **Compile for** drop-down at the bottom. Select *MultiMarkdown – Latex*. That will change the list of **Compilation options** on the left to show only the ones relevant for that compilation route. Now we can look at each of those options[1] in turn to see what can be adjusted.

Contents

The *Contents* pane lets us to choose which documents of our draft text to include in the compile. The folder drop-down lets you view the contents of folders, or you can view the whole draft. You then click the tick box to select a file, or opt-click a box to select all the files in a folder. As we're compiling through MultiMarkdown we can ignore the `As is' and `Page break before' boxes – they do nothing for us.

Before you leave the *Contents* option make sure the folder drop-down is set to the highest level you want to compile (which will usually the *Draft* folder); if it isn't, you'll only get the contents of the folder which is selected.

Separators

At last, an easy one. In the *Separators* pane set all of the options to *Empty line*.

Formatting

When compiling to MultiMarkdown we use the formatting pane (Figure 4.1 to select the document levels we want to include in the compile. For each of the levels in the binder we need to make sure the *Title* and *Text* boxes are both ticked.

If Level 1 or lower aren't showing just click the plus button (marked in red on Figure 4.1) to add all the levels you need.

The *Level settings* should just show `# Title #` with `Main text` below. There's nothing to change there.

[1] Chapter 23 of the Scrivener manual covers compilation in detail: we're just looking at the important options for ScML.

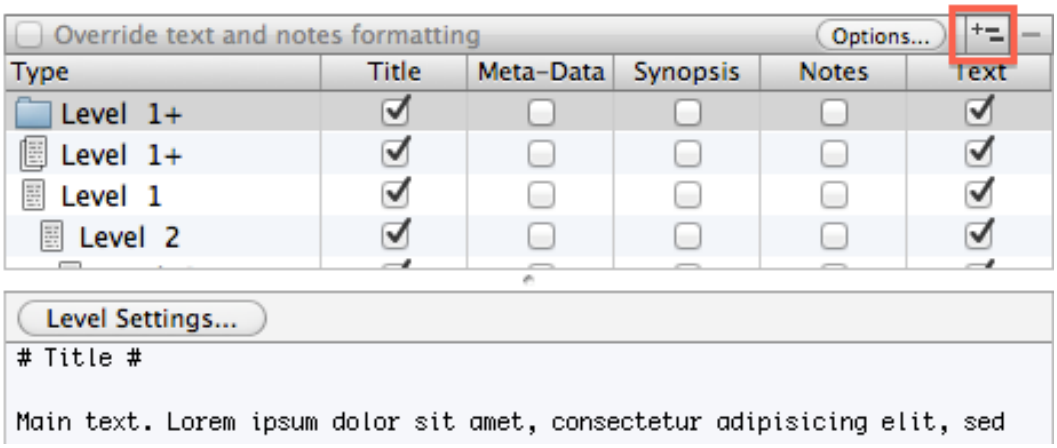

Figure 4.1: The formatting pane

Layout

Nothing to see here, move along there.

Transformations

The *Transformations* pane controls a few document wide changes, such as changing all smart quotes to straight quotes. Pick the ones you need.

Replacements

The *Replacements* pane is potentially a very powerful tool for ScML. You can use it to set up replacements of words or characters which take place during compile. I haven't done much with it yet, but I can see possibilities for simplifying the markup in Scrivener.

For example, pasting an equation from MathType gives the standard Latex syntax: `\[equation\]`, while MultiMarkdown needs double backslashes (and that extra space): `\\[ equation \\]`. If you only ever wanted to use the standard Latex maths delimiters when writing you could set up two replacements:

```
\[ to \\[

\] to  \\]
```

When you compiled that would add the extra \ without you having to worry about it.

Note that you can have both project and preset replacements:

- project replacements apply to any compile from the project;
- preset replacements apply only to compilations using a named compiled preset of compilation options (see section 23.4.1 of the Scrivener manual for a discussion of compile presets).

Preset replacements enable you to apply different sets of replacements for compiles of the same document: you might use one set if compiling to MultiMarkdown and a different set for compiling to HTML or .mobi.

Statistics

Another one ignore.

Footnotes

If you need to stop comments, annotations and footnotes going through to the compile do that here.

Meta-Data

Metadata is the key to managing MultiMarkdown. If you're doing MultiMarkdown by hand you need to have a metadata section in your document, but in Scrivener we can put all our metadata in this Compile option pane (Figure 4.2).

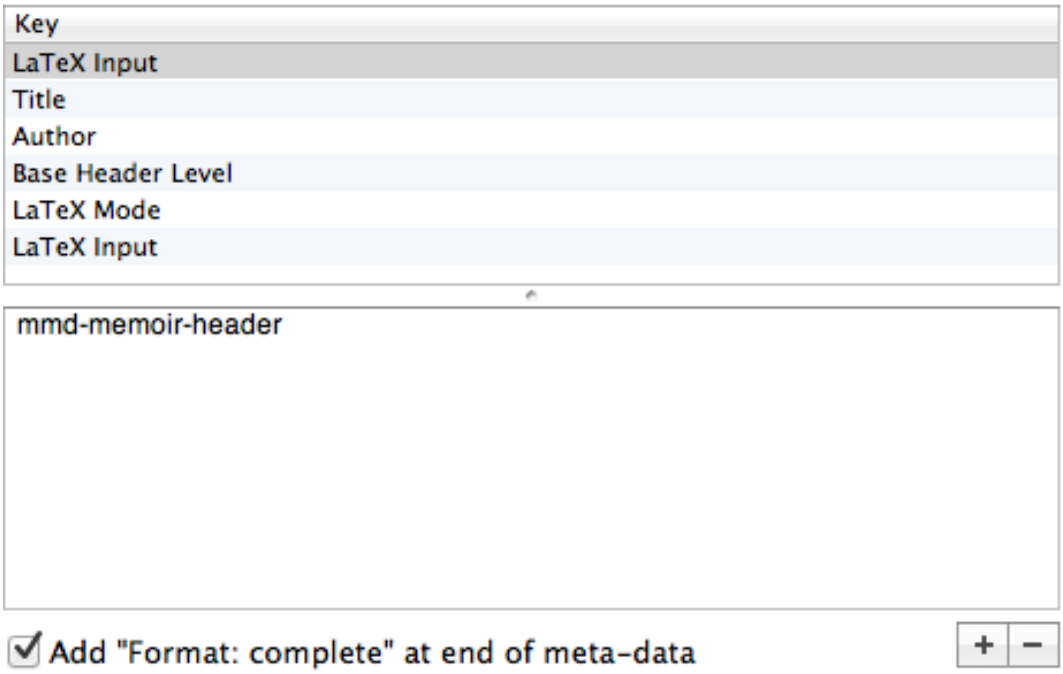

Figure 4.2: The Metadata compile pane

The main pieces of metadata are:

- Latex input: specifies a tex file which will be used during typesetting using a latex `\input{}` command. These files contain the Latex commands for document layout and typesetting options.
- Author: your name goes here.
- Title: your document name goes here.
- Base header level: Latex has a fixed order of headers, from Part, which is level one, through Chapter (level two), section, sub-section, sub-sub-section. This setting determines how the highest level of your document will be treated.

  I've set my base header level to two, so the highest division I get is chapter (with no parts). If I'd set the base header level to one then what is currently Chapter 4 Compiling would be Part IV Compiling.

- Latex mode: MultiMarkdown can use two Latex classes (document types), memoir or beamer. Memoir is for page layout PDF documents, beamer for presentations.
- Latex footer: a special case of the Latex input. This file is called at the end of the tex document during typesetting. It handles end matter like glossaries and bibliographies.

The metadata settings for this document are shown in Table 4.1. You'll see there are two Latex input entries, one for mmd-memoir-header which comes before the Author, Title and Base header level, and one which comes after. That order is important for proper processing.

Table 4.1: Metadata for the toad's guide

Metadata	Values
Latex input	mmd-memoir-header
Title	A toad's guide to using Scrivener – MultiMarkDown – Latex
Author	Huw Evans
Base Header Level	2
Latex mode	memoir
Latex input	mmd-memoir-begin-doc
Latex footer	mmd-memoir-footer

In Scrivener we add metadata items by clicking the plus button, then editing the title in the list and entering the content in the text box. We can re-order the metadata items by dragging them up and down the list.

4.2 Pressing the button

With the compile options all set it's time to press the Compile button. Suddenly stuff happens:

1. The **Export** dialogue appears, asking for a file name and location. Give it what it wants and click **Export**.

   If you've compiled the document before Scrivener will offer the last used file name and location, then ask you if you want to replace it. Heck yes is the answer.

2. Scrivener now compiles the text. While this is happening the progress bar at bottom left fills steadily blue.

3. Now MultiMarkdown picks up the compiled text (the progress bar clears) and processes it. While this is happening the progress bar fills with blue and white spiral.

4. The **Export** dialogue retracts and we're done.

Usually, compiling goes smoothly: mis-typed links and fouled-up syntax tend to make themselves known in the Latex typesetting or weird effects in the finished PDF. Now and again though there are glitches: you'll get a warning dialogue and the compile fails. The only solution is to go back, try and fix the issue then try again. What little advice I have to give is in section 5.6.

4.3 Where'd it go?

The next challenge is finding your compiled file. Exactly where it turns up depends on whether there were linked images or not:

- if there were no linked images then you'll find the compiled file in the specified directory, named `filename.tex`.

- if there were linked images then there will be a folder `filename.tex`, and inside that will be the exported images and the compiled file, named `filename.tex`.

Chapter 5

Typesetting (TexShop)

When we get to TexShop we don't compile the document we *typeset* it. It's a mists of time thing, going back to the days when documents were typeset and not tidied up into PDFs. But before we can get on with typesetting we have to know a little bit about how tex files are structured.

5.1 A bit about Latex

Tex files consist of three main sections, a beginning, a middle and an end. The beginning – the header – contains the general instructions for page layout and fonts, as well as the instructions for assembling tables of contents. The tex files produced by MultiMarkdown don't have that header information built in; instead it is supplied at the point of typesetting by calls to other documents. This makes it comparatively easy to customise you Latex output[1].

The starting point for our the Latex header data is the tex files named in the metadata back at the Scrivener compile dialogue. There are two files:

- mmd-memoir-header.tex
- mmd-memoir-begin.tex

The files are called using the standard Latex `input{}` command, which means the Tex engine works through the tex file until it gets to an `input{}` then pulls in the referenced file and works through that, then jumps back to the main file, and so on. So for this document, the first few lines of the tex file are:

```
\input{mmd-memoir-header}
\def\mytitle{A toad's guide to …}
\def\myauthor{Huw Evans}
\def\latexmode{memoir}
\input{mmd-memoir-begin-doc}
\def\format{complete}
\chapter{Introduction}
\label{introduction}
```

We can see the calls to those two files, as well as the outcome of some of the other metadata attributes, such as the title and author. We'll have to look at the insides of those tex files a little later, but for now, we just need to know enough to get the document typeset the first time.

[1] If your need for customisation goes beyond that provided by the tex documents you can revert to the MMD 2 method and hack about with XSLT files. But this is where I leave you.

5.2 Setting up for typesetting

There's one thing we have to do to get things ready for typesetting. We have to make sure that the text files called by those `input{}` commands are sitting in the same folder as the tex document itself.

Unfortunately, that isn't just two files, because those two files include a few `input{}` commands themselves. We'll be picking the chain apart in a later section, so the easiest thing when you're first experimenting with ScML is to make sure all the files which came as part of the MultiMarkdown Latex support package are in the same folder as your tex file. That way they *have* to be in the right place.

With the files in the right place you can launch TexShop and open your tex file. It'll open in a dull-looking window with a toolbar at the top. There's a **Typeset** button on the toolbar. Press it.

TexShop grinds into action. You'll know that because the **Console** window opens and starts to fill with *stuff*. What you want to see at this point is that the *stuff* keeps scrolling by until it gets to a point where it says `Output written on …` and a new window opens showing your new PDF.

And that's it. Except that you'll now want to make it look the way you want it to look and not the default look. Which means we need to come back to those `input{}` calls and get a better idea of what is going on. We're also going to have to talk about packages.

Of course, there's always the possibility that your typesetting didn't result in a PDF and the Console may have stalled, in that case, have a look at section 5.6 for a few hints.

5.3 Those other files

As you'll find out once you start tinkering, the chain of `input{}` in those first two files goes like this:

- mmd-memoir-header
  - mmd-memoir-setup
    - mmd-memoir-layout--8.5x11
    - mmd-memoir-packages
    - mmd-default-metadata
- mmd-memoir-begin-doc
  - mmd-title
  - mmd-memoir-copyright

Each of the files does a little something for you, and each of them can be customised. For example, because I use A4 paper I've replaced the *mmd-memoir-layout--8.5x11* with my own *mmd-memoir-layout-A4*. I've also added to the packages listed in *mmd-memoir-packages* to call up the ones I want.

There are similar chains if you want to produce stand alone *articles* rather than the longer memoir.

One of the benefits of having the chains, rather than two big files, is that you can link to them from different starting points, allowing you to make a few major decisions on how you want stuff to appear, and then re-use that time and again. The downside is that you can get tangled (which is why I wrote out those lists in the first place).

5.4 Packages

From here on in it's down to you to make Latex work for you. But before I finish there are a couple of points to make on customising. The first is packages.

Packages are Latex's way of bolting additional functionality to the standard set: they are the go-faster stripes of typesetting. If you come across things you can't do with standard Latex there's probably a package which will let you do it. Some packages handle layouts, others give you SI units or chemical

symbols. There are packages for most things. The Tex User Group archive[2] is a good place to start looking, but it's worth noting that many of the packages come with the MacTex distribution.

5.5 Other choices

Latex has many flavours. People eventually find out what works for them. One of my personal preferences is to use the Xetex/Xelatex[3] derivative for typesetting. It's accessible as one of the options in TexShop. The main benefit is that you can use all your standard OS X fonts, without being restricted to the smaller set of Latex specific fonts.

5.6 Compile and typeset woes

The basic rule for happy compiling and typesetting is to compile and typeset little and often. Do not wait until the whole hundred and fifty thousand word epic is finished. Process each chapter as you go. Sure, some of the cross-references will fail as they refer to sections as yet unwritten or outside the current compile selection, but you'll catch the stupid mistakes early and fix them.

Identifying MultiMarkdown compile problems is a pain, as you are trying to find a mis-placed bracket or caret in the middle of all your text. The best approach here is to re-compile with progressively fewer documents in Scrivener, until you identify the one which is causing the failure. Once you get it down to one document it's then relatively easy to find your mistake: even if you end up with the extreme solution of taking out all the MultiMarkdown and starting again.

Finding Latex typesetting problems is a different issue. The console will flag up errors as it goes:

- layout problems like over-full boxes or missing references will appear as warnings but typesetting will carry on.
- issues like missing images or incorrect maths layout will stall the typesetting, but a few returns in the console will re-start the process. This is often down to a `[` instead of a `{` (or the other way round) or a missing curly brace in maths or SI units.
- some errors are severe enough to bring the whole process to a juddering halt.

For the first two cases you can go back to the console after typesetting has finished and look at the reporting line numbers to find and solve the problem. For the third, it may be an unfinished maths expression has produced an unresolvable horror: those are hardest problems to track, but often, solving earlier errors resolves later ones. Again, little and often is the rule for avoiding huge, untraceable issues.

The final set of issues only appear when you go through the PDF and find that things don't look right. Once again, I find it's maths expressions give the most problem, followed by tables and links. And usually it's my fault for missing off a `!` or not putting the head-body divider line in a table, or misspelling a cross-reference. You need to go through the document with the Scrivener project open, working between the preview, the tex document and Scrivener. Often I'll find the mistake in the pdf or console, test a fix in the tex document, then carry that change back to Scrivener.

At the risk of being boring, little and often. And the more complex it is, the littler and the oftener. We have to accept that compiling is not a once-and-for-all process, but an iterative one which gives a better and better document.

[2] http://www.tug.org/ctan.html
[3] http://tug.org/xetex/

Chapter 6

Extras

I have mentioned one or two other tools which I use around this workflow, but this is the consolidated list. Beyond buying their software, I have no financial connection with the developers or distributors of these programs.

- **MultiMarkdown Composer**: a stand-alone editor for MultiMarkdown which is useful for testing tricky bits of code. It comes from the creator of MultiMarkdown, Fletcher Penney. £6.99 through the App Store.
- **QuickCursor**: a handy bolt on which quickly copies text from the open application into a chosen application, useful for working between Scrivener and MultiMarkdown Composer. Hog Bay Software. £2.99 through the App Store.
- **MathType**: a visual equation editor which can produce Latex code for pasting into Scrivener. Design Science[1]. $99 from their web store.
- **yEd**: I've just found this one. A flow-chart creator from yworks [2]. And it's free.
- **pdfsam**: PDF splitter and merger. When I need to break a PDF up into chapters, usually to accompany distance-learning modules, this does the job. You can split at page or bookmark level, making it easy to slice a PDF into chapters. PDFsam.org[3]. Also free.

[1] http://www.dessci.com/en/
[2] ttp://www.yworks.com/en/products_yed_about.html
[3] http://www.pdfsam.org/