The EBW Electric Book Workflow

A guide to creating high-quality books and ebooks with open-source tools

Arthur Attwell

The Electric Book Workflow

A template for creating high-quality books and ebooks with open-source tools

Arthur Attwell

EBW

A Guide to the Electric Book Workflow

ISBN (Print): 978-1-928313-13-7
ISBN (Digital download): 978-1-928313-14-4

Contents

Preface

I developed this workflow at Electric Book Works for producing Bettercare books, before using it for other imprints and for client work at Fire and Lion. Thanks to Emma Attwell, Tarryn-Anne Anderson, Julia Norrish, Barbara Attwell and Karen Lilje for being enthusiastic early users.

This guide to the workflow is a work in progress.

Arthur

Introduction

At Electric Book Works, we needed a system that is easy for non-technical people to edit books in, includes great version control, produces books fast (no InDesign layout except for cover design), and instantly produces HTML we can use for the web, ebooks, apps, and print. By print, we mean high-end books you buy in a store, not just 'save as PDF'. We didn't want to be locked into any proprietary tools, either: we needed to use open technologies to ensure our content was future-proof.

So we put this workflow together:

1. We store a book's master files in plain text (formatted as markdown).
2. We turn that plain text into clean HTML instantly (using Jekyll).
3. We apply different stylesheets to that HTML to get beautiful web versions, ebooks and print PDFs in minutes (using Sigil and PrinceXML).
4. Everything is stored securely with state-of-the-art version-control (using GitHub or BitBucket).

Markdown is so simple that non-technical people can learn how to create and edit it in under an hour. (They can even work

directly on the master files online using Prose.)

Jekyll is great at generating clean HTML in flexible ways, and has a big, active development community and committed project owners. Is also includes kramdown and Sass, which are critical to creating new book designs quickly.

While PrinceXML is proprietary software, it's our favourite implementation of the open standards for CSS Paged Media. You might also try Antenna House. And as more open-source tools (e.g. WeasyPrint) support CSS Paged Media more fully, we'll have more options here.

Alternatives

There are several digital-first book-publishing systems around.

Some are also based on markdown, like ours. For example:

- Gitbook IO
- Penflip
- LeanPub (which uses Markua)
- Phil Schatz's viewer.

PressBooks, which is built on Wordpress, is a superb, affordable service.

And if you're a serious publishing outfit, have a look at O'Reilly Atlas.

OERPub Editor is a web-based editor for non-technical people to create EPUB3 HTML that includes maths. As a

Javascript editor it runs in the browser and saves content in EPUB3 structure to a GitHub repo. Neat.

For converting HTML/CSS to print PDF, we use PrinceXML, which is proprietary. Other alternatives, which we haven't tested or don't yet support all the page-layout features we need, include:

- Antenna House (proprietary, mature)
- WeasyPrint (open-source, in active development)
- Vivliostyle (young, apparently open-source, but also commercial)
- PDFReactor (proprietary)
- DocRaptor (a cloud-service implementation of PrinceXML)

Setting up

Technical overview

To set up the workflow, you still have to have some technical expertise. Once it's set up, non-technical editorial team members with a couple of hours training (taught or self-taught) can add and

edit books in it.

The technical team members who run the workflow need to be familiar with a few concepts and tools:

- **HTML and CSS**: the fundamental building blocks of almost all digital content.
- **Markdown**: a simple, plain-text shortcut for creating HTML. (The original Markdown syntax reference is the easiest intro to basic markdown. We use a markdown variant called kramdown, because it's GitHub's default and it supports attributes like classes.)
- **Sass**: a way to create complex CSS from simple rules.
- **Jekyll**: software that turns markdown and Sass into HTML and CSS. (To learn about Jekyll, start here. If you're installing it on Windows, you'll also need this guide.)
- **Git**: software for tracking a team's changes and syncing them with a remote server. We like to use GitHub and Bitbucket as our remotes.
- **Sigil**: an open-source epub editor, where we quickly assemble ebooks.
- **PrinceXML**: an app for creating PDFs from HTML and CSS (Prince is the only proprietary part of this stack).

Example: Bettercare

Bettercare publishes nursing textbooks. The team uses the workflow to keep a book's master content in markdown

files, structured for Jekyll, on GitHub. For instance, Bettercare's books are stored here: https://github.com/bettercarehealth/bettercare

> Note: Bettercare is a great example of a big publishing project using the workflow. However, since it was our first full implementation, it uses an early, outdated version of the workflow.

The workflow uses the kramdown syntax for markdown, in part because that's what GitHub uses and largely because it has specific features we need, such as classes (with 'inline attribute lists' or IALs).

For Bettercare's open-source books, we let GitHub Pages publish the static HTML output, which it does automatically, also with Jekyll. For instance, our staging site for Bettercare content is here: http://bettercarehealth.github.io/bettercare/

Bettercare then copies the output HTML to a separate, production site.

When we use this workflow for closed content, we don't use GitHub Pages, and store the content in private GitHub or Bitbucket repos instead.

If you click through to a book chapter available on the web, you'll see the HTML we get from kramdown is very neat. For example, view source here: http://bettercarehealth.github.io/bettercare/nc/nc-1.html

The key to simple HTML is in carefully mapping a

book's features to ordinary HTML elements. That way, we need only a few classes, and can easily use the same HTML with simple stylesheets for the web, apps, epub, and print output. And our HTML content remains readable in readers and low-bandwidth browsers with low CSS support.

Finally, to turn our HTML into print PDFs, we use PrinceXML. And we use Sigil to put our HTML into EPUB2-valid epubs. This way, we can create print PDFs or epubs in a matter of minutes.

Folder (repo) structure

A workflow folder (often tracked in Git as a repo) usually contains a series of related books. Its folders and files follow the standard Jekyll structure: in the root are `_include`, `_layouts` and `css` folders, and `_config.yml` and `index` files. We then put each book's content in its own folder.

> Pro tip: You could also store several series in one repo, each series with its own set of Jekyll files, and a single `_prose.yml` configuration in the root folder for all series subfolders. This is only useful if you don't need a live staging site or previews with GitHub Pages, since each

> Jekyll setup must be in its own repo for GitHub Pages to work out of the box.

Using the template

The `template` folder is a ready-to-use workflow folder for making books. In the `template` folder, there are several folders and files:

- `book` : book content files
- `css` : book design files
- `_includes` : snippets of HTML for Jekyll (you won't have to open this folder)
- `_layouts` : instructions to Jekyll on how to assemble those snippets (no need to open this either)
- `_data` : files containing book metadata
- `_config.yml` : a file for setting configuration options for your collection of books
- `_prose.yml` : configuration settings for using prose.io for online book editing (generally, you won't have to edit this)
- `index.md` : content for the home page of your collection when served as a website.

The `book` folder is an example book with a minimum number of sample files in it.

The files in the `css` folder handle the design of your books. It contains a theme that we call 'Classic'. A theme is a collection of CSS (Sass, technically) files that define a book's design. We hope that in future we and others might design other themes, though Classic is extremely versatile already. For most book design, you'll only have to edit the variables in your own copies of `print.scss`, `web.scss` and `epub.scss`.

> Technical note: If you're familiar with Sass, you'll know that Jekyll's will convert these `.scss` files into finished `.css` files. Sass, saved in `.scss` files, is to CSS what markdown is to HTML: an efficient way to write that lets software do the hard work of creating finished code.

Creating a new book

To create a new book in a new series:

1. The `template` folder can hold one book or many, like a collection of books that share similar metadata or features (e.g. they're all by the same publisher). Make a copy of `template` and rename it for your collection. E.g. `my-sci-fi`.
2. Rename the `book` folder with a short folder-name version of your book's title. Use only lowercase letters and no spaces. If you're creating more than one book, make a copy of this

folder for each one. Each book gets its own folder. The name of the folder is important: you will use this folder name as the book's 'slug', a sort of human-readable identifier within a series folder.

3. Inside a book's folder, add a markdown file for each piece of your book, e.g. one file per chapter. Our template contains files we consider minimum requirements for most books: a cover, a title page, a copyright page, a contents page, and a text file.
4. Inside each book's folder, store images in the `images` folder. Add a `cover.jpg` image of your book's front cover there, too.
5. In the `css` folder, make copies of `print.scss`, `web.scss` and `epub.scss` and rename them for each book (e.g. `print-space-potatoes.scss`).
6. Inside `my-sci-fi`, open and edit these three files:
   - `_config.yml`: Edit the values there for your series. The comments will guide you.
   - `index.md`: Replace our template text with your own. Usually, at least a link to each book is useful, e.g. `[Space Potatoes](space-potatoes)`.
   - `README.md`: Replace our template text with any notes your collaborators might need to know about your series. (The README file is usually only read in the context of editing the files in your folder/repo.)
7. In `_data`, make a copy of the `book.yml` file to edit. Give the file the same name as your book's slug. In this example, that would be `my-sci-fi.yml`. In that file, fill in as much metadata as you can. (Your markdown files and HTML snippets will draw on that metadata, using Liquid tags, when constructing your book.)

Creating book content

Each markdown file in `my-sci-fi` is a part of a book, such as a table of contents or a chapter. In each file's YAML header (the info between `---` s at the top) we specify the book-part's `title` and the book-part's `style` to use for that part. The `style` specifies what kind of book-part it is, such as a `title-page` or `chapter`.

> Technical note: the `style` YAML sets the class attribute of the output HTML's `<body>` element. We use that class to control CSS and page structure.

When you create your book, we recommend following these conventions for file naming and `style` settings:

Book section	Sample file	Style in YAML
Front cover (for the ebook)	`0-0-cover.md`	`cover`
Title page	`0-1-titlepage.md`	`title-page`
Copyright page	`0-2-copyright.md`	`copyright-page`
Table of contents	`0-3-contents.md`	`contents-page`
Acknowledgements	`0-4-acknowledgements.md`	`frontmatter`
A first chapter	`1.md`	`chapter`
A second chapter	`2.md`	`chapter`

If you don't set the `style`, the page will default to `style: chapter`. So you actually don't need to ever set `style: chapter` in a YAML header.

Page layouts we've designed for in the Classic theme include:

- `index` for the home page of a collection
- `cover` for a front cover, which will appear in ebook editions
- `halftitle-page` for a book's halftitle page
- `previous-publications-page` for a book's list of the author's previous publications
- `title-page` for a book's title page
- `copyright-page` for the copyright or imprint page
- `contents-page` for the book's table of contents
- `dedication-page` for a dedication page
- `epigraph-page` for an epigraph page
- `frontispiece-page` for a frontispiece page
- `frontmatter` for other prelim pages not accounted for otherwise
- `chapter` for a book's default chapter page (and the global default)

You can also invent your own page styles, and use them in your custom CSS instead of these, though you may get unexpected results if you've been relying on CSS for existing styles like `chapter`.

> Tip: If, in your web output, you don't want the navigation (nav bar and footer) on a page, such as the collection's

index page, add `layout: min` to the document's YAML header. The `min` layout does not include a nav-bar and footer.

Set first pages

Many books have two 'page ones':

1. the half-title or title page and,
2. if the prelims have roman-numeral page numbers, the first chapter.

You should specify those pages so that Prince knows where to start numbering.

Why? Well, for example, in print output if you use `frontmatter` on a book-part, by default it will have roman-numeral page numbers. When the first `chapter` starts, it will have decimal page numbers. However, the page numbering will be consecutive from roman through decimal. That is, it will run 'ix, x, 11, 12'. You reset the numbering to 1 at the start of the first `chapter` to avoid this.

You reset page numbering by adding the class `page-1` to the first block-level element on the relevant page.

You can do this in two ways:

1. If a markdown document starts at 'page one', add the class to the `style` YAML header. E.g.

```
---
```

```
title: Half-title page
style: halftitle-page page-1
---
```

And at the first chapter:

```
---
title: Chapter One
style: chapter page-1
---
```

Remember that `chapter` is the default, so you normally don't have to specify it. *But* if you want to *add* a class in addition to `chapter`, you must specify both classes. This is because, if you were to use `style: page-1` in a YAML header, the class `page-1` would override the default `style: chapter`, not add to it.

2. Add the `page-1` class to the first block-level element in the chapter by adding the tag `{:.page-1}` in the line immediately after it. (Note: for this to work, the element must *not* have a CSS float applied to it.)

File naming

Name each book's markdown files in perfectly alphabetical order. We recommend using a numbering system, where prelims (frontmatter) files start with a 0, e.g. `0-1-titlepage.md`, `0-2-copyright.md`, and chapter files are numbered for their chapter number, e.g. `01.md`, `02.md`, and so on. The alphabetical

order makes it easy to see the documents in the right order at all times, and it makes ordering outputted HTML files easy when dropping them into Prince for PDF output.

> Note: We recommend using leading zeros in file-name numbers – that is, `02.md` rather than `2.md` – because that sorts correctly in most file browsers. Otherwise, some file browsers will sort `10.md` before `2.md`. In the rare event that you have over 99 chapters, use two leading zeros: `001.md`.

The `images` folder

Alongside the content files in a book's folder is an `images` folder, for images that belong to that book only.

A book's folder should only ever need to contain markdown files and images. If you're embedding other kinds of media you could add folders for that alongside `images`. We don't recommend sharing images or media between books, in case you want to move a book from one repo to another later. (So, for example, copy the publisher logo into each book's `images` folder separately.)

Markdown

This guide explains how to edit text for the workflow. On its own, it's also a handy guide to markdown in general.

If you get your markdown right, the workflow can zap out an almost-print-ready PDF, website or ebook from your files in a matter of minutes. That is, we go straight from edited manuscript to page proofs – no typesetting!

To edit for the workflow, you need to have a rough idea how HTML works, and you need to learn markdown. If you're new to HTML and markdown, don't worry: it's much easier than you think.

How HTML works

HTML is a computer language. It's a way for us to mark up or tag text, so that a computer knows how to display it beautifully to humans. That is, software (most commonly a web browser) reads the HTML code and *renders* it as nice-looking, readable text and images.

In HTML, each piece of content (e.g. a paragraph) starts and ends with a tag that a computer can recognise. Tags are always in elbow brackets, like this paragraph tag: `<p>`.

```
<p>
This text is tagged in HTML as a paragraph.
</p>
```

The slash 'closes' the tag.

The `<p>` tag is one of about a hundred possible tags, maybe ten of which you'll see in a book, such as tags for headings (e.g. `<h1>`, `<h2>`), bold and italic, lists, images, tables, and blockquotes.

But not all paragraphs are the same: there are opening paragraphs, pull-quote paragraphs, blurb paragraphs and more. If I want to tell the computer what *kind* of paragraph it is, I can put it in a special class. Say, the 'blurb' class of paragraph. To write this in HTML, I would make the tag `<p class="blurb">`. Unlike its limited set of specific tags, HTML classes are infinite, because we can make up class names as we need them.

HTML is very simple and very powerful (which is why it's the language behind almost every web page). We can mark up everything in a book using HTML tags and classes. In our workflow, we store books in HTML, with each book part (title page, contents page, chapter, etc.) in its own HTML file.

Then, to create print-ready PDFs, websites and ebooks from that HTML, we combine the files with CSS stylesheets. CSS is another computer language that defines design: font sizes, indentation, colour, etc. A CSS file will say 'make all headings bold', for instance, or 'indent blockquotes by 1em'.

If we combine a book's HTML with a stylesheet designed for print, we can get a print-ready PDF. If we combine the same HTML with a stylesheet for the web, we get the book as a website. This way, we only ever store the book's content once, and use different stylesheets to generate alternative editions from that single source.

Unfortunately HTML tags are very, very time consuming to type by hand. But we have a shortcut! It's called markdown, and it's amazing.

How markdown works

Markdown is just plain text, neatly structured. Plain text as in `.txt` files. It's so neatly structured that software can work out, just from the structure of your text, what HTML you intended, and convert it instantly to finished HTML, tags and all. No way! Yes way.

Markdown was invented by two very smart people in 2004: John Gruber and Aaron Swartz. Since their work, others have added new features to markdown. So today there are many variations of markdown. We use a variation called kramdown.

As you work with markdown, you'll get to know it really well. That can take as little as a few hours. And hopefully you'll come to love it as much as we do. To do that, you'll need to keep the kramdown syntax reference handy. For now, though, we'll explain the most common, important things you'll need to edit simple books.

To quickly test any basic markdown yourself, go to http://kramdown.herokuapp.com. Type markdown in the left, and see the rendered HTML on the right.

Paragraphs

This is easy: a paragraph is lines of text separated from any other text by an empty line. Markdown:

You can even have line breaks in a paragraph, and markdown will just ignore them (or, rather, replace them with spaces). Markdown's looking for an empty line before it ends a paragraph.

```
This is a paragraph.

This is another
paragraph.
```

Result:
This is a paragraph.
This is another paragraph.

Headings

You can use up to six levels of heading, from level one to level six. To make a heading, just put one or more hash signs, `#`, and a space, before the heading. For a first-level heading, use one hash sign; for a second-level heading, use two; and so on. Like with paragraphs, separate the heading from everything else with an empty line space.

```
## This is a second-level heading

This is a paragraph.
```

Italics

Just put `*` before and after the italicised words: `is it really *this* easy?`. It's really *this* easy.

Bold

Just put two `*`s before and after the bold text: `good for **shouting**`. **Shouting.**

Lists

There are two kinds of lists: bulleted and numbered. For a bulleted list, start each line with a `*` and a tab. For a numbered list, start each line with a number, a full stop and a tab (you can use any number, because the software will always create an HTML list that starts with 1; but it's best to use 1 or the actual numbers you intend, just to make things neat). Markdown:

Tip: If you use bold in a first-level heading (e.g. `# **Chapter 1** Lost at Sea`), our default typography will put that bold text onto its own line, set off from the heading. This is useful for chapter numbers that should look separate from the chapter title, but in the underlying HTML need to be part of the heading, for instance when software generates tables of contents.

```
1.        Apples
2.        Oranges
3.        Pears

*        Apples
*        Oranges
*        Pears
```

Result:

1. Apples
2. Oranges
3. Pears

- Apples
- Oranges
- Pears

Simple tables

You can create simple tables in markdown. You can make them really neat, or you can make them really efficient. Markdown:

```
| Fruit   | Quantity |
|---------|----------|
| Apples  | 2        |
| Oranges | 5        |
| Pears   | 0        |

| Fruit | Quantity
|---|---
| Apples | 2
| Oranges | 5
| Pears | 0
```

Result:

Fruit	Quantity
Apples	2

Fruit	Quantity
Oranges	5
Pears	0

Fruit	Quantity
Apples	2
Oranges	5
Pears	0

To make more complex tables with merged or individually styled cells, you have to use actual HTML table markup. See the Tables chapter for more detail.

Note the minimum three hyphens in the lines that creates the border between the table head and table body. (Kramdown doesn't require three, but GitHub previews do, so it's best practice.)

Blockquotes

Remember how old email programs put a `>` at the start of each line when you hit 'Reply'? Ah, `>` means blockquote. Start each line with `>` and a space to make text a blockquote. Markdown:

```
> This is text in a blockquote.
```

Result:

> This is text in a blockquote.

In our workflow we use blockquote HTML for more than just quotations, though. We also use it for figures and poetry. We'll come back to this later.

Hyperlinks

If you want your final HTML to include a clickable link:

- surround the text you want clickable with `[` and `]`
- put the URL it should point to between `(` and `)` immediately afterwards.

Markdown:

```
To learn more, [click here](http://google.com).
```

Result:
To learn more, click here.

Images

To place an image, you include a line telling the computer where to find the image file. This looks like a hyperlink, but with a `!` at the start of the line. Inside the square brackets, you include a brief description of the image (this is useful to screen readers for the visually impaired).

```
![A dog chasing a bus.](images/dogbus.jpg)
```

In the parentheses above, the `images/` part says that the

`dogbus.jpg` image is in the `images` folder, which is alongside the markdown file. In our workflow, we always put images in the `images` folder like this.

See the Images chapter for more detail.

Endnotes

Endnotes in our workflow appear at the end of a document (that is, a web page or a book chapter).[1]

- put a `[^1]` where the footnote reference should appear (the `1` there can be any numbers or letters, and should be different for each footnote in a document);
- anywhere in the document (we recommend after the paragraph containing the footnote reference), put `[^1]: Your footnote text here.`.

We'll explain how to create notes on the side or bottom of a page (footnotes) later when we talk about classes.

(By the way, endnote references are `<sup>` in kramdown's HTML, not unicode numbers like [2]. We style for these and for `<sub>`, as in H_2O.)

See the Notes chapter for more detail.

Definition lists

A definition list is a list of definitions, not surprisingly. Think of a dictionary. A definition list comprises one or more entries, and each entry has a headword and a definition. Even if you're not

editing a dictionary, you might need to create a short glossary or define a word or two at some point. To create a definition entry in markdown, put the headword on its own line, and the definition on the next, after a colon and a tab. Markdown:

```
Editor
:	Someone who spends more time learning new tricks
than making money.
```

Result:

Editor
: Someone who spends more time learning new tricks than making money.

To create a whole list of definition entries, just put one of these after the other, with a line space between them.

Using class tags

We're going to get a bit more advanced now. Get some tea.

As we mentioned earlier, sometimes we have to tell our software what *kind* of paragraph or list or blockquote we want. These *kinds* of text are called *classes*. To give something a class, we add a kramdown tag in curly braces, with a colon, and a dot

before the class name. Markdown:

```
This paragraph should be in a box.
{:.box}
```

Result:

> This paragraph should be in a box.

For the resulting HTML to actually appear in a box, when we turn your markdown into a finished book, we'll have to combine it with a CSS stylesheet that includes formatting instructions for the `box` class. If you're just editing in markdown, you don't have to think about the stylesheets. Our default stylesheets include designs for `box` as well as several other classes. (Which we'll list in a moment.)

But if your book needs classes that aren't already in our workflow's default stylesheets, you'll need to ask a CSS-savvy designer to write you some CSS rules for your new class, and to put these in a child stylesheet.

Inventing classes

If you invent new classes, make sure you name them for their semantic *purpose*, not their appearance. For instance, `important-tip` is a better class name than `shaded-bold`.

Also, class names should always be lowercase and have no spaces.

In addition to boxes, our default stylesheets include lots of other classes for common book features.

Block and inline elements

Classes can be applied to two kinds of element: *block* elements and *inline* elements.

A block element is anything that should (in print or on screen) start and end with a line break, like a paragraph or a list. An inline element is anything that appears inside a line of running text, like bold and italic. Most of our pre-designed classes are for either block or inline elements, and sometimes both.

When you apply a tag to a block-level element in kramdown, you put the tag on the line immediately following the element.

When you apply a tag to an inline element, mark off the text you're tagging with asterisks (`*`), as you would italics, and put your tag immediately after the closing `*`, on the same line. E.g. `*Make this small-caps.*{:.smallcaps}`.

To see all the classes that our Classic theme supports, see the Sup-

ported classes section.

Editing reflowable text

When you're working in markdown, you're creating text that might reflow in an infinite number of ways on screens and onto page layouts. This changes the way you edit, because nothing is static.

For instance, deliberate use of non-breaking spaces is very useful:

- To prevent ellipses falling after a line break, add a non-breaking space (` `) before ellipses.
- To prevent dashes falling after a line break, add a non-breaking space (` `) before the dash: `the en-dash – as I've explained – is tricky.`
- To prevent line breaks inside big numbers (e.g. at the space in 40 000), between numbers and units (e.g. in 3 pm, 24 May), or between adjectives and their units (e.g. in Grade 2), use a non-breaking space (` `) in the space:
  - `40 000`
  - `3 pm`
  - `Grade 2`

1. In kramdown syntax, unfortunately, endnotes are called footnotes; so it's easy to confuse them. In book parlance, there is a big

difference between footnotes and endnotes.

Themes and stylesheets

Our template includes one default design, or theme, which we called 'classic'. This theme is a set of stylesheets for the three main output formats: the web (`web.css`), ebooks (`epub.css`) and print (`print.css`).

A theme is made of component parts:

- variables like default fonts and page size
- book elements like running heads and boxes.

You set your series variables and choose which components to include in the Sass files in `/css` : `web.scss` , `print.scss` and `epub.scss` .

When Jekyll converts your markdown to HTML, it will read your Sass files and automatically assemble finished CSS.

To change stying for a specific book in a series, make a copy of the stylesheet(s) you need to change (`web.scss` , `print.scss` and/or `epub.scss`), and name it sensibly (e.g. `web-scifi.scss`).

Then add the file name(s) to that book's details in `_config.yml` . For instance, if you created a style for

`web-scifi.scss` but will use the default print and epub styles for the book, you'll this line to your configuration for that book:

```
stylesheet-web: "web-scifi.css"
```

Supported classes

Our Classic theme includes styling for a range of classes. You can apply these to elements in markdown.

Formatting

Use these classes in your markdown to create specific formatting effects.

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Bibliography list	`.bibliography`	Block	Styles a list as a bibliography, for instance at the end of an academic book.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Box	`.box`	Block	Puts the element in a box, to set it off from the rest of the text.	No
Chapter number	`.chapter-number`	Block	Used for a chapter number before a chapter heading. (See the tip at Bold in the chapter on Markdown for another way to handle chapter numbers.	No
Dedication	`.dedication`	Block	A dedication, for instance at the start of a book or chapter	No
Epigraph source	`.epigraph-source`	Block	The person to whom the epigraph is attributed.	No
Epigraph	`.epigraph`	Block	An epigraph at the start of a book or chapter.	No
Figure	`.figure`	Block	A figure is an image with a caption. Read about how to manage them properly in our workflow guide.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Figure: extra small	`.x-small`	Block	Add to the `.figure` tag, e.g. `{:.figure .x-small}`.	No
Figure: fixed position	`.fixed`	Block	Add to the `.figure` tag, e.g. `{:.figure .fixed}`. For figures that must keep their position in the text flow, and must not float to the top of the page.	No
Figure: large	`.large`	Block	Add to the `.figure` tag, e.g. `{:.figure .large}`.	No
Figure: medium	`.medium`	Block	Add to the `.figure` tag, e.g. `{:.figure .medium}`.	No
Figure: small	`.small`	Block	Add to the `.figure` tag, e.g. `{:.figure .small}`.	No
First paragraph	`.first`	Block	For any paragraph that starts a new set of paragraphs, flush left and with a gap above it.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Float to top	`.float-top`	Block	Floats the element to the top of its page. Useful for boxes. Applies to print output only.	Yes
Float to bottom	`.float-bottom`	Block	Floats the element to the bottom of its page. Useful for boxes. Applies to print output only.	Yes
Footnote	`.sidenote .bottom`	Block or inline	When you add `.bottom` to `.sidenote`, the note appears at the foot of the page in print output. It remains on the side on screens. (Also see the chapter 'Footnotes, endnotes and sidenotes'.)	No
Fraction	`.fractions`	Block or inline	If your font supports it, converts characters like `1/2` into fraction characters.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Glossary	`.glossary`	Block	Use this after the last entry in a series of definition lists to define the entire list of definitions as a glossary.	No
Hide from print	`.non-printing`	Block or inline	Hides the element from print output. Useful for things like clickable buttons, which are only intended for screens, not paper.	No
Keep together	`.keep-together`	Block	Prevents an element from breaking across pages. (E.g. you want to keep a short list on the same page.)	No
Keep with next	`.keep-with-next`	Block	Prevents a page break between this element and the next one.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Logo image	`.logo`	Block	Used for making images small, especially for small logos in text like on acknowledgements pages.	No
Page break after	`.page-break-after`	Block	Creates a page break after the element.	No
Page break after, end of book	`.page-break-after-right`	Block	When applied to the very last element in the book, ensures a blank verso for an even-numbered page extent.	No
Page break before	`.page-break-before`	Block	Starts its element on a new page.	No
Page break: allow	`.allow-break`	Block	Allows an element to break over a page where the default styles would normally prevent that. Apply the class to the parent element.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Page numbering restart	`.page-1`	Block	Restarts page numbering from 1. Can be added to the first block element on a page, or to the YAML header, in addition to the main style, e.g. `style: halftitle-page page-1` or `style: chapter page-1`. Recommended for any document that starts a book interior (e.g. title page), to retain correct pagination when creating a PDF ebook with a front cover	No
Poetry	`.verse`	Block	Designing poetry is tricky and important. Read about how to manage this in our workflow guide.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Pull quote	`.pullquote`	Block	Displays a paragraph as a pull quote.	No
Sidenote	`.sidenote`	Block or inline	A sidenote appears in a sidebar to the right of the text.	No
Small caps (lowercase only)	`.smallcaps`	Block or inline	If your font supports proper small-caps glyphs. Only affects the lowercase letters.	No
Small caps throughout	`.allsmallcaps`	Block or inline	If your font supports proper small-caps glyphs, all characters are small caps.	No
Source after a quotation	`.source`	Block	The name and/or title of the source for a preceding quotation.	No
Table caption	`.table-caption`	Block	Add `{:.table-caption}` in the line immediately after a table caption. Table captions must always appear above tables, not after them.	No

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Table caption	`.table-caption`	Block	Use for the caption above a table. (Table captions should not appear after tables, only before.)	No
Title page: author	`.title-page-author`	Block	The book's author(s) on the title page.	No
Title page: logo	`.title-page-logo`	Block	A logo, as an image, on the title page.	No
Title page: subtitle	`.title-page-subtitle`	Block	The book's subtitle on the title page.	No
Title page: title	`.title-page-title`	Block	The book's title on the title page.	No
Tracking: tighten	`.tighten-1` to `.tighten-50`	Block or inline	Each increment tightens the space between letters by 0.001em (1/1000 of a em). *Affects print output only.*	Yes
Tracking: loosen	`.loosen-1` to `.loosen-50`	Block or inline	Each increment loosens the space between letters by 0.001em (1/1000 of a em). *Affects print output only.*	Yes

Feature	Workflow class	Block or inline	Explanation	Supports edition suffix
Valediction	`.valediction`	Block	Used for the sign-off at the end of a letter, preface or foreword.	No

Reserved classes

You may also need to create your own classes for other uses. If you do, avoid using the same already-supported class names above. You should also avoid using the following ones, which are reserved for specific structural elements.

Class name	Reserved for
`index`	The home page of a collection, used for the `style` value in file YAML headers
`cover`	A front cover, which will appear in ebook editions, used for the `style` value in file YAML headers
`halftitle-page`	A book's halftitle page, used for the `style` value in file YAML headers
`previous-publications-page`	A book's list of the author's previous publications, used for the `style` value in file YAML headers

Class name	Reserved for
`title-page`	A book's title page, used for the `style` value in file YAML headers
`copyright-page`	copyright or imprint page, used for the `style` value in file YAML headers
`contents-page`	A book's table of contents, used for the `style` value in file YAML headers
`dedication-page`	A dedication page, used for the `style` value in file YAML headers
`epigraph-page`	An epigraph page, used for the `style` value in file YAML headers
`frontispiece-page`	A frontispiece page, used for the `style` value in file YAML headers
`frontmatter`	For other prelim pages not accounted for otherwise, used for the `style` value in file YAML headers
`chapter`	A book's default chapter page (and the global default), used for the `style` value in file YAML headers

The edition suffix

If you want to produce more than one print edition of a book from the same source file, you can't use the same classes that

affect text-flow – like `.tighten-1`, for instance – in both editions, because the text will flow differently in each edition.

Our workflow has a way to manage that. In the print CSS file, you can specify an edition suffix. For instance, if you're producing a schools edition of a book, you might make your suffix `-schools-edn`. That suffix will be appended to the end of certain class names for that stylesheet. The default `.tighten-1` class will become `.tighten-1-schools-edn` in your final print CSS.

(It's a good idea to start a suffix with a hyphen and use all lowercase letters, to keep your output CSS neat. Never use spaces.)

Only some classes are affected – see the table above for which ones. The most important are the classes used for tightening and loosening letter-spacing, which are mostly used to control widows and orphans in print layout.

In your markdown, then, you'd use `{:.tighten-1-schools-edn}` instead of `{:.tighten-1}`, and that class will then only have an effect on your schools edition. If you had another edition, say a large-print edition with a `-large-print` suffix, you'd use a `{:.tighten-1-large-print}` tag in the markdown. These would match the classes automatically generated in each edition's CSS.

Of course, one element can carry both classes. For instance, you might end up with a paragraph tagged with `{:.tighten-1-schools-edn .tighten-1-large-print}`. That paragraph would then be tightened in both print layouts.

Deprecated classes

Early versions of the EBW used the following classes, which are no longer supported:

- `shrink`
- `tighten`, `tight`, `x-tight`, `xx-tight`, `xxx-tight`
- `loosen`, `loose`, `x-loose`

Tables of contents

A table of contents should be its own markdown file containing a list of the book's parts. Each list item should be a link to a location in the book – ideally a fragment identifier to ensure that print page references are always accurate. Here is a simple example:

```
---
title: Contents
style: contents-page
---

# Contents

1. [Chapter 1](1.html#chapter-1)
1. [Chapter 2](2.html#chapter-2)
```

For more advanced markup and styling, you may need to do more. In this example, each element is tagged.

```
---
title: Contents
```

```
style: contents-page
---

# Contents

1.
[*Acknowledgements*{:.toc-chapter-title}](0-5-acknowledgements.html#ackn
1.
[*Preface*{:.toc-chapter-title}](text.html#preface){:.frontmatter-refere
1. [*1*{:.toc-chapter-number} *Early Days in
Mavambe*{:.toc-chapter-title}](text.html#early-days-in-mavambe)
1. [*2*{:.toc-chapter-number} *Baragwanath Hospital and
Beyond*{:.toc-chapter-title}](text.html#baragwanath-hospital-and-beyond)
```

Note the `.frontmatter-reference` in particular: our default styles format that page number according to the style you set for the `$frontmatter-reference-style` variable in your print stylesheet. The other classes in this example are for custom formatting.

> Tip: We recommend using a numbered list, even though out default styles hide the numbers. This is to make it easier to convert your TOC to EPUB3, which requires that the `nav` element on your contents page contains only an ordered list.

Images

Adding images in markdown

We use standard markdown to embed images:

```
![The image alt text](images/filename.svg)
```

Most of our images are figures. That is, they include an image followed by a caption. We put these together in a `<blockquote>` element with a `.figure` class. We can then control placement by styling the `<blockquote>`.

The reason we use a blockquote is that it lets us keep images and their captions together. A `<figure>` element would be better HTML, but it won't validate in EPUB2, and can't be created with kramdown.

Here's an example of markdown for a figure:

```
> ![Figure 2-A: The Ballard scoring method](images/
fig-2-A.svg)
>
> Figure 2-A: The Ballard scoring method
{:.figure}
```

Every line (except the `{:.figure}` class tag at the end) starts with a `>` and a space. These wrap the figure (image and caption) in a `blockquote` element.

The first line is the image reference. It consists of:

- an exclamation mark telling markdown that we're placing an image
- the `alt` attribute in square brackets
- the path to the image file.

The third line is the figure caption, followed by the kramdown tag `{:.figure}`, which lets our stylesheets format the `blockquote` as a figure. (For instance, preventing a page break between the image and the caption in print.)

If your image has no caption, just skip the empty line and

caption line:

```
> ![Figure 2-A: The Ballard scoring method](images/
fig-2-A.svg)
{:.figure}
```

If it's important to you that the image isn't in a blockquote, and there is no caption, you can use:

```
![Figure 2-A: The Ballard scoring method](images/fig-2-A.svg)
{:.figure}
```

Always check print output (putting the HTML Jekyll creates through Prince with `print.css`) to be sure you're getting what you intend.

Alternative image sets

There are often good reasons for producing books with different sets of images. For instance, one edition may have colour images and another black-and-white. Or your print edition might call for high-res images, but you want low-res ones for the ebook.

By default, your markdown looks for images in the book's `images` folder. You can set your config file to switch to an alternative set of images (in a subfolder of `images`) when needed.

In your markdown, instead of using the `images` path (e.g. `images/filename.jpg`), use this tag: `{{page.image-folder}}`.

For example:

```
![]({{page.image-folder}}/filename.jpg)
```

Your default `image-folder` is set to `images` in `_config.yml`. You can change that for each book as needed. For instance, you might create a subfolder for high-res images in `images` called `hi-res`. In `_config.yml` you can then add a line to set that subfolder as the default images folder.

```
image-folder: images/hi-res
```

Whenever you build your HTML, you then switch which set of images to use by commenting out the image-folder setting(s) you don't want. For example, when creating the high-res edition, in `_config.yml` you might comment out the default image path so that the HTML uses the high-res images:

```
#image-folder: images
image-folder: images/hi-res
```

Remember that your images in your alternative images folder must have exactly the same file names as your default image set. The HTML will be looking for the same file names in a different folder.

To always use a specific image file for any given image, irrespective of the `images-folder` config, simply hard-code your image path in markdown – that is, without using the `{{page.image-folder}}` tag. For example, for a given image you might specify the default images folder `![](images/filename.jpg)` or a specific subfolder `![](images/nb/`

`filename.jpg)`.

Image placement

You may need to control how an image is sized and placed on the page – especially in print – depending on its detail or aspect ratio and nearby images or other elements. You do this by adding the class tag to the line after the `>` lines. (This applies a class to the blockquote in HTML.) You have these options:

- `.x-small` limits the image height. In print, to 30mm.
- `.small` limits the image height. In print, to 45mm.
- `.medium` limits the image height. In print, to 65mm, which allows two figures with shortish captions to fit on a page.
- `.large` fills the width and most of a printed page, up to 150mm tall. Try to put these images at the end of a section, because they cause a page break.
- `.fixed` keeps the figure in its place in the text flow, and will not float it to the top or bottom of a page. For instance, when an image must appear in a step-by-step list of instructions.

You add these classes to the `{:.figure} tag like this:

```
{:.figure .small}
{:.figure .fixed}
```

and so on. You can combine size and placement classes like this, too:

```
{:.figure .fixed .small}
```

CSS tip: If you're having trouble with SVGs having space around them, in your CSS make sure you set the height of the `img` element. SVGs are inline elements by default, and will add white space around them.

Preparing images

- Wherever possible, convert images to SVG so that they scale beautifully but also remain small in file-size for web use.
- Ensure that raster images, or raster/bitmap elements in SVG, are high-res enough for printing (e.g. 300dpi at full size).
- Embed images placed in your SVG image, don't just link them.
- Create a JPG version of every SVG image with the same file name (e.g. `bear.svg` and `bear.jpg`). You'll need the JPG fall-back for EPUB. (We recommend JPG over GIF or PNG as a general default. One reason is that transparency seems like a good idea until your end-user switches their e-reader to 'night mode', and your black line-art disappears into the background.)
- Save images in the book's `images` folder.

Here's our most common workflow for converting images to SVG:

- If the image was created in InDesign (e.g. a flowchart made of InDesign frames): open in InDesign, group the frames that make up the image, copy, and paste into a new Illustrator file.

Adjust Illustrator file artboards as necessary, then save as SVG.

- If the image was created in Photoshop or other raster format: open the original, copy into Illustrator. Live trace the image. (We mostly use the 'Detailed Illustration' preset.) Save as SVG. You can also use the trace function in Inkscape instead of Illustrator.
- If you save SVG from Adobe Illustrator (and possibly other creators, too), choose to convert type to outlines. (For us, the main reason for this is that PrinceXML gives unpredictable results with type in SVG. But it's probably safest generally to convert type to outlines for consistent output.)

Recommended image sizes

We like to use these settings where possible:

- Default width: 115mm
- Aspect ratios: 4:3 (portrait or landscape), a closer ratio, or square. Images at wider ratios (e.g. 16:9) than 4:3 make layout more difficult.
- Therefore, maximum height is 150mm. (That's very slightly less than a 4:3 height:width ratio.)

Using Illustrator? Different SVG editors treat image size differently. For instance, a 2-inch-wide image in Illustrator will appear 1.6 inches wide in Prince and Inkscape. Why? Because when creating the SVG's XML, Illustrator includes its dimensions in pixels, and *assumes a 72dpi resolution*, where Prince and Inkscape follow the W3C SVG spec and assume 90dpi. As a result,

images coming out of Illustrator always appear 80% of their intended size. So, if you're creating images in Illustrator, set your image sizes to 125% of what you intend to appear in the book. That means:

- default width 115mm × 125% = 143.75mm
- max height (at 4:3) = 190mm

Check out Adobe's guidance on saving SVGs.

Resolution

- For SVG images, the editor you use will determine underlying resolution. Illustrator uses 72dpi, and Inkscape 90dpi. We favour and assume 90dpi, but can rescale SVG images with our stylesheets just in case.
- For JPGs, our default resolution should be 200dpi and image quality of 80 ('very high' in Adobe presets). This allows for reasonable print quality while keeping file sizes sensible for web delivery. The higher resolution also allows ebook users to zoom in for more detail.
- To get a 200dpi JPG that is 115 mm wide, the image must be 906 pixels wide. (115mm is 4.53 inches, which contains 906 pixels at 200 pixels per inch, aka 200 dpi.)
- Try to keep JPG file sizes below 127KB: Amazon Kindle may automatically downsample images above that, and it's better if you control the downsampling for quality than let their servers do it. However, for raster-only images (e.g. x-rays or photos) if a larger size is required for acceptable print quality then larger is fine.

Image styles

We like these approaches to artwork, where possible:

- Default style: Black and white line art, with average 1mm line thickness.
- Use shades of grey only where needed, and as few shades as possible.
- Use the same font and size for all labels. (We like Source Sans Pro at 10pt on 115mm-wide images, 11pt on 145mm-wide images (see note above on using Illustrator).
- Fit artboards to artwork bounds; there must be no white space around the art in an image. (We control space with styling.) Since you're creating images to a specific size, you need to **expand artwork to fit the artboard**, *not* fit artboards to artwork bounds, which would make your whole image smaller.

If you use live trace to create art from a raster source, you must clean up the file to remove unnecessary fills that add to file size but do little for clarity.

Settings:

- Default file format: SVG
- Convert type to outlines (the alternative to embed and subset fonts doesn't work reliably in print output currently)
- Raster elements embedded, not linked
- Transparent background

Where images *must* be raster (e.g. x-rays, photos), they should follow the sizing constraints above and be saved as jpg (since Amazon Kindle only uses JPG or GIF, avoid PNG or other

formats). Save as RGB.

Where labels are added to a raster image, the image should be saved as SVG with an embedded raster image. Labels and other text should *not* be rasterised.

If you're creating images from InDesign originals using Illustrator, a suggested workflow:

- If the image was created in InDesign (e.g. a flowchart made of InDesign frames): open in InDesign, group the frames that make up the image, copy, and paste into a new Illustrator file. Adjust Illustrator file artboards as necessary, then save as SVG.
- If the image was created in Photoshop or other raster format: open the original, copy into Illustrator. Live trace the image. I mostly used the 'Detailed Illustration' preset. Save as SVG.
- For filenaming, use the convention 1-2.svg, as in chapter-figure.svg. For skills workshops images, that might be 1E-B.svg for workshop 1E, figure B. All the images go in an images folder inside the folder with the markdown files.
- If you save SVG from Adobe Illustrator (and possibly other creators, too), choose to convert type to outlines. Currently, PrinceXML does not support fonts in type in SVG reliably.
- To save a little more on file size, also convert all strokes to fills.

Image file sizes

If you SVGs seem big, read up on optimising SVGs, and/or (if you're comfortable using Python scripts) run your SVGs through Scour.

Cover images

Add the front-cover image to the book's `images` folder. Ensure colour settings are RGB and the DPI is set to 72. We recommend creating the image in three sizes:

- `cover.jpg` : 960px high (in keeping with epub best practice these are just under 1000px on their longest side)
- `cover-thumb.jpg` : 300px wide
- `cover-large.jpg` : 2000px high

The first is mandatory. The thumbnail and large images are for your convenience. For instance, when uploading a book to Amazon Kindle, you must provide a cover image this large.

Embedding video

You can include any iframe in markdown to embed a video. We've created a simple way to embed YouTube videos and have them look consistent across your web version.

- Find the video's YouTube ID: a code in the URL that looks like this: `RRV-9Jf0eI0`
- In the markdown, put the code between these two tags: `{% include youtube-start.html %}` and `{% include youtube-end.html %}`. Those tags will insert the iframe HTML that works best for the Learning Station.
- Our default iframe code gives the iframe a `.non-printing` class, so that the video won't appear in the printed or PDF book.
- If you need a heading, caption or any other text related to the video, remember to add `{:.non-printing}` to them, so that they also do not appear in the book.

Here's a full example:

```
## A video
{:.non-printing}
```

```
{% include youtube-start.html %}RRV-9Jf0eI0{% include
youtube-end.html %}
```

Note that this only works with YouTube. If you're embedding from any other service, instead of using our `include` tags:

- use their standard embed iframe
- try to select a width of around 850 px
- add `style="max-width: 100%;"` and `class="non-printing"` to the iframe tag.

Footnotes, endnotes and sidenotes

Our default theme provides three options for notes.

Footnotes appear at the end of a document (web page or book chapter). In book parlance, they are therefore actually endnotes, but we call them footnotes because that's what kramdown calls them. To create them in markdown, follow the kramdown syntax for footnotes:

- put a `[^1]` where the footnote reference should appear (the `1` there can be any numbers or letters, and should be different for each footnote in a document);
- anywhere in the document (we recommend after the paragraph containing the footnote reference), put `[^1]: Your footnote text here.`.

Sidenotes appear in a box to the right of the text. On wide screens, they float far right of the text. On narrower screens, the text wraps around them. In print, the text wraps around them, too. To create a sidenote, put a `*` at the start of the sidenote text and `*{:.sidenote}` at the end (with no spaces). (Technically,

you're creating an `<em>` span with a kramdown IAL.)

In print, you can put **sidenotes at the bottom of the page**. By adding `.bottom` to the `{:.sidenote}` tag, your sidenote sits at the bottom of the page rather than on the right with text wrap, replicating a traditional footnote. So the markdown looks like this: `*This is a sidenote at the bottom of the page in print.*{:.sidenote .bottom}`. On screen, these are just regular sidenotes.

Tables

Kramdown (and most markdown variants) can only handle very simple tables. For these you can create the Markdown layout manually.

For complex tables – anything with merged cells, for instance – you must create an HTML table and paste that HTML (from `<table>` to `</table>`) into your markdown files. We use a WYSIWYG tool like Dreamweaver, which lets us paste a table from a formatted source like Word or LibreOffice, and then clean up the HTML easily before pasting the whole `<table>` element into our markdown document.

Add `{:.table-caption}` in the line immediately after a table caption. Kramdown uses this to apply the class `table-caption` to the paragraph. In our print output, this lets `print.css` avoid a page break after the caption, before the table. (According to publishing best-practice, table captions must always appear above tables, not after them.)

Markdown-tables tools

Sometimes we use Senseful's online tool to create markdown tables quickly:

- Click and drag over some cells in the InDesign table (not the header row). Then Ctrl+A to select the whole table.
- Ctrl+C to copy, then paste into a blank spreadsheet.
- Select all the relevant cells in your spreadsheet, and copy. The table text is now on your clipboard, with the cells separated by tabs.
- Paste into the online Format Text as Table Input field.
- Click 'Create Table'. (The default settings are usually fine. Play with them only if you need to.)
- Copy the Output and paste it into your markdown file.
-

The Senseful tool starts some table borders with + where kramdown needs a	. Manually change the starting + in any row with a	.

Indexes

Introduction

An index (the long list with the page numbers at the end of a book) is an important part of many non-fiction print books (many ebooks don't include indexes, and instead rely on ebook app's search function). Traditionally, indexes are created by professional indexers, who read the final page proofs and compile a list of entries and page numbers manually. They do not only add words to the index, but entire concepts. For instance, a good indexer will include an entry for 'democracy' in an index if the

concept is discussed, even if the word 'democracy' is never used. This is why good indexes cannot be generated by computers.

In this workflow, the challenge is in managing page numbers: our workflow allows you to reflow the content into new formats, so page numbers are never quite fixed. Well, you might decide that a particular format's pages are fixed enough for allow for manual page numbers in an index, but then you cannot reuse that index in another format.

So here we'll describe how to create an index where the page numbers are dynamic. That is, the index is a list of concepts that point to a particular point in the book. The page numbers in the index (and clickable hyperlinks in an ebook version) are then generated on the fly whenever you output to PDF. This also means you can index early in the production process, such as during editing, rather than waiting for pages to be finalised.

> Note: Be careful not to confuse an index in a book with the `index` page of a website. When talking about websites, the `index` page is the home page of a directory. So never give your reference index the file name `index` (e.g. `index.md` or `index.html`). To avoid confusion, in our code we refer to the `reference-index` when we mean a book's index.

To create an index you have to do two things:

1. In the text, tag the words that their index entries will point to.
2. Create an index document with a list of entries.

How to tag indexed words

To tag a word in the text, we make it a link, and we give that link:

- a target that points to the index
- a class (`.indexed`) to control what it looks like
- an ID (e.g. `#democracy-1`) to uniquely identify it.

The link target should point to the index entry for that word. This way, clicking the word in the text will take you to the right place in the index. And clicking the page reference in the index will take you to the word in the text.

Here's an example of the link in the text:

```
Late that night, [Bob](reference-index.html#bob-1){:.indexed
#bob-1} realised the key was in his pocket.
```

And then when Bob appears later in the book:

```
Eventually, [Bob](reference-index.html#bob-2){:.indexed
#bob-2} called her to confess.
```

Note that the ID must be unique for every instance of Bob.

How to create the index list

Then in the index itself, you create a list of entries. After each

entry, you add links to each instance of that entry you've tagged in the text. And you give each link the ID that you've pointed to in your tagged word's link target.

> Tech tip: To make it easy for us to manage, we use the same ID for the tagged word and the index entry. You don't have to us the same ID. For instance, your tagged-word's ID might be `#text-bob-1` and your index entry's link `#index-bob-1`. If like us you use the same ID in both cases, remember that your index must be in a separate file from your text, since IDs must be unique within each file.

Here is what your markdown for the index might look like. Here we've included examples of sub-entries, and the tag to use for the entire list to style it as an index: `{:.reference-index}`.

```
* Alice
[1](1.html#alice-1){:#alice-1}
[2](3.html#alice-2){:#alice-2}
* Bob
[1](1.html#bob-1)
[2](9.html#bob-2)
* key
[1](4.html#key-1)
    - private
    [1](4.html#key-private-1)
    - public
    [1](4.html#key-public-1)
{:.reference-index}
```

In this example, we've used numbers as the link text. On screen, these will stay numbers, e.g.:

Alice 1, 2

The stylesheet will add the commas between entries (so you could globally replace with semicolons or otherwise). In print output (using PrinceXML), the stylesheet will replace those numbers with page references.

Future development

We hope to find more concise ways to create indexes from tags in running text in future. We'd also like to align this work the IDPF's recommendations on indexes in ebooks.

Poetry

Basic poetry markdown

Encoding poetry can be tricky. Usually, poetry in HTML is structured by tagging each stanza as a paragraph, with line breaks after each line. You can do this by adding markdown line breaks (with double spaces or `\\` at the end of each line) and tagging the paragraph with `{:.verse}`. However, this structure meaks it impossible to have browsers, ereaders and PDF engines correctly indent runover lines (because there is no `nth-line` selector in CSS, unless you resort to a Javascript method that will bloat your code and won't run on many ereaders).

> Tech note: Some text and code editors (e.g. Atom) strip out spaces at the ends of lines automatically. So use \ for line breaks, not double-spaces.

We prefer another approach: the poem is an unordered list (`ul`) and each line (including each blank line between stanzas) is a list item (`li`). We just hide any list markers (bullets) with `list-style-type: none` . This way, we can control indents on runover lines. This is a non-semantic use of HTML, since a poem is technically not a list. But it's a healthy hack with universal browser and ereader support.

Our convention is to mark each line of a stanza with a hyphen `-` , and tag the list with `{:.verse}` :

```
-        I wandered lonely as a cloud
-        That floats on high o'er vales and hills,
-        When all at once I saw a crowd,
-        A host, of golden daffodils;
-        Beside the lake, beneath the trees,
-        Fluttering and dancing in the breeze.
{:.verse}
```

We can also indent individual lines, where the poet wanted indents, by tagging individual list items.

```
-        Alas for man! day after day may rise,
-        {:.indent-3}Night may shade his thankless head,
-        He sees no God in the bright, morning skies
```

```
-        {:.indent-3}He sings no praises from his guarded
bed.
{:.verse}
```

The `2` in `{:.indent-2}` refers to the number of em spaces to indent by. Our CSS allows for indents from 1 (`{:.indent-1}`) to 30 (`{:.indent-30}`).

(Big gaps between words in a line must be created with ` ` in the poem text.)

Advanced poetry markdown

But wait, there's more! Best practice for poetry layout is that – in print – a poem should be centered on its longest line. That is *not* centering the lines of poetry, but placing the left-justified poem in the horizontal middle of the page. Put another way, the poem should be indented till its longest line is centered on the page.

To achieve this, put the entire poem, including its title, in a blockquote, by adding `>` to the start of each line. Tag the whole blockquote as {:.verse}, too. Finally, decide how wide you want the poem to be in multiple of 10 per cent. That is, if you reckon this poem's longest line reaches across 90 per cent of the page, use `.width-90`.

```
> -        ### To One Who Has Been Long in City Pent
> -        To one who has been long in city pent,
```

```
> -        {:.indent-2}'Tis very sweet to look into the fair
> -        {:.indent-2}And open face of heaven,—to breathe a
prayer
> -        Full in the smile of the blue firmament.
> -        Who is more happy, when, with heart's content,
> -        {:.indent-2}Fatigued he sinks into some pleasant
lair
> -        {:.indent-2}Of wavy grass, and reads a debonair
> -        And gentle tale of love and languishment?
> -
> -        Returning home at evening, with an ear
> -        {:.indent-2}Catching the notes of Philomel,—an eye
> -        Watching the sailing cloudlet's bright career,
> -        {:.indent-2}He mourns that day so soon has glided
by:
> -        E'en like the passage of an angel's tear
> -        {:.indent-2}That falls through the clear ether
silently.
> {:.verse}
{:.verse .width-90}
```

In verse structured as a list like this, our CSS preserves white space. That is, if you type, say, three spaces you get three spaces. Normally, HTML collapses multiple spaces into one – which is great *except* when you want to deliberately use extra spaces, as some poets do. However, this doesn't work at the start of lines, where markdown strips leading spaces. There you must use HTML space entities (like ` `) or our indent tags explained above.

However, the `whitespace: pre-wrap` CSS we use for this is

not currently supported on Kindle. If that's important, it's best to stick to using HTML space entities like ` `.

Page numbers

In print output in the Classic theme, the YAML header's `style` setting determines how the pages are numbered:

- all book parts set to `style: frontmatter` have roman-numeral page numbers;
- all book parts set to `style: chapter` (or, since that's the default fallback, all those without a `style` setting) have decimal page numbers;
- other book parts (e.g. `style: copyright-page`) have no page numbers (nor any other headers and footers).

So if you use `frontmatter` on a book-part (e.g. for a preface, foreword or acknowledgements section), by default it will have roman-numeral page numbers. And when the first `chapter` starts, that chapter will have decimal page numbers.

However, the page numbering will increment consecutively from roman to decimal, unless you tell PrinceXML to start from 1 at the first `chapter` . That is, 'ix, x, 11, 12'. To reset the numbering to 1 at the start of the relevant `chapter` , you have two options:

1. Add the page class `.page-1` to the first element (e.g. the first heading) of the first chapter. In markdown, you do that with the tag `{:.page-1}` in the line immediately after the heading.

This only works if that element is not set to `float` in the CSS. (That is, the element must appear in the normal flow of text.)

2. Add `page-1` to the `style` YAML setting, *in addition* to specifying `chapter`, not overriding it. That is, the first chapter in your book should have the YAML style set as: `style: chapter page-1`.

Converting from InDesign

Suggested steps

This is what we do when we convert one of our textbooks from a traditional InDesign workflow to markdown for this workflow. You'll probably develop your own process for turning existing books into markdown. Maybe this will help.

1. Open the InDesign file and copy all the text.
2. Paste the text with formatting into your text editor.
3. Search and replace (S&R) all line breaks with double line

breaks:
    - Tick 'Regular expression' (because you're using the regex `\n` to mean 'line break', not actually searching for the characters 'slash' and 'lowercase en').
    - Find `\n`
    - Replace with `\n\n`
4. Comparing to a laid-out, up-to-date version of the book, mark all headings with markdown's heading hashes (#) according to their heading level. E.g.:
    - `h1` = `#`
    - `h2` = `##`
    - `h3` = `###`, and so on.
5. At the same time, you may want to manually create Markdown lists using `*` for bullets and `1.`, `2.`, `3.` etc. for numbered lists. Note that list indents can get complicated, so check previous chapters and test your markdown-to-HTML conversion when you hit a tricky one (e.g. a note inside a bullet list nested inside a numbered list).
6. Add Markdown image references. See the images section below for detail on how best to do this.
7. Search for every instance of italic, bold and bold-italic, and manually mark these in markdown with asterisks: in markdown, `*one asterisk*` is *italic*, `**two**` is **bold**, and `***three***` is ***bold-italic***. Remember that in some fonts, italic and bold may have different names in InDesign, like 'oblique' and 'black'.
8. Look out for special characters, especially degree symbols (°), superscripts and subscripts. It's best to search the InDesign document (search by style and basic character formats, e.g. 'Position' for superscript and subscript) for these instances so

you don't miss any. Most superscripts and subscripts in InDesign and similar are created by formatting normal text. In text-only, there is no formatting, so you should use the unicode character for each superscript or subscript character. E.g. when typing the symbol for oxygen, O_2, the subscript 2 is ₂, unicode character U+2082. To type these characters, you may need special software (e.g. for Windows, Google unicodeinput.exe), or copy-paste from an online reference. In Windows, you can also find symbols in Character Map, e.g. search in Character Map for 'Subscript Two'.

Search-and-replace in InDesign

InDesign is great for prepping text you've received in InDesign or Word, because it has very powerful search-and-replace. In particular, it's worth getting to know GREP search-and-replace. The learning curve is steep, but your conversions will be much faster once you've mastered GREP. Here are some GREP learning resources.

For instance, if you want to convert all italics to markdown:

1. In the Find dialog, choose GREP.
2. Click in the Format box and set the font style to Italic.
3. In the Find box, search for `(.+)`.
4. In the Replace box, replace with `*$1*`.

Remember that search-and-replace is rarely flawless, so proceed

with caution. For instance, regarding italics: in markdown, it's very important that the starting `*` and ending `*` that mark the italics are right up against the words they wrap. I.e.:

- Correct: `I want to *emphasise* this.`
- Incorrect: `I want to *emphasise *this.`

However, many (human) editors often mark the spaces on either side of a word as italic, probably because clicking and dragging with a mouse can be inaccurate, and because double-clicking a word often highlights the word as well as the space after it. This means that when we automate 'search for italics and wrap with `*`', we end up wrapping the space into the markdown as well. Also, sometimes spaces in the manuscript are italic, even though the words on either side of the space are not (this is probably caused by editing or pasting phrases that were once italic). In Word and InDesign, you'd never spot those italic spaces by eye. But in the markdown, you end up with this:

```
I went to* *the shops.
```

Markdown reads that is meaning that there is an actual asterisk after 'to', and italics start from 'the' and continue (at least) to the end of the paragraph:

I went to* *the shops.*

These problems have to be fixed with a combo of clever searches and manual spotting.

The same thing happens with bold, but luckily bold is usually used more sparingly than italic, and is therefore easier to fix manually.

Print output

We use PrinceXML to turn Jekyll's HTML into beautiful, print-ready book files. We haven't found anything as good as Prince, so we reckon it's worth its price tag. And you can use the trial version to get your print output perfect before committing to the price. So our CSS files for print are designed specifically for Prince.

If you want to learn about using CSS to control print output using Prince, this is a great tutorial.

Creating PDF files with the Prince GUI

1. Find your book's HTML files in your `_site` folder. (Remember to run Jekyll locally to generate the latest version; if you're getting your HTML from a GitHub repo's `_site` folder, trust that the last contributor synced up-to-date, reliable HTML generated by their local Jekyll instance.)
2. Drag the HTML files into Prince. Make sure they're in the right order. Only include the files you need in print. For instance, you'll usually leave out `index.html`. You'll usually only include `cover.html` when creating a PDF ebook.
3. Tick 'Merge all…' and **specify an output file** location and name. Do not let Prince output to your `_site` folder. (If you skip this, Prince will output to your `_site` folder, which will cause permissions issues when you want to modify or delete the file, because Jekyll owns the `_site` folder and can overwrite it, and because Git will try to commit and sync the output PDF, which you probably don't want.)
4. Prince will find your book's print CSS, which you specified in `_config.yml`. If you want a PDF with a completely different CSS file, either:
    - change it in `_config.yml` and restart Jekyll to update your files with the new config, or
    - leave it blank (`stylesheet-print: ""`) and manually add the output CSS to Prince.

5. To remove crop marks or to remove bleed entirely, add one of these CSS files to Prince (they're also in `_site/css`):
   - `print-no-crop-marks.css` (removes the crop marks, but leaves the bleed as is)
   - `print-no-bleed.css` (removes the bleed, and any crop marks with it).
6. Click Convert.

   Tech detail: Our default HTML `head` includes links to both the screen stylesheet and the print stylesheet specified in `_config.yml`. Prince knows which one to use because they're specified with `media="screen"` and `media="print"`.

Creating PDFs with Prince from the command line

For general instructions, see the official PrinceXML user guide. We like to use this workflow:

1. In each book's folder (alongside the markdown) we keep a file called `print-list`. The `print-list` file is just a list of the HTML files that must be included in the print PDF. E.g.:

```
0-1-titlepage.html
0-2-copyright.html
```

```
0-3-contents.html
1.html
2.html
3.html
```

2. At your command prompt, navigate to the book's folder in `_site`. (With the right settings, you can launch your command line app in the right folder from a file explorer in Windows, Mac and Linux.)
3. At the prompt, type `prince -l print-list -o /temp/filename.pdf`.

> In that last step, `/temp` is an existing folder at the root of your drive (e.g. `C:/temp` if you're working on your C drive in Windows) and `filename` is a sensible file name. If you just used `filename.pdf` without the folder path, the PDF would be saved to the same folder as the HTML in `_site`. We don't recommend this because Jekyll will overwrite that whole folder if you build new HTML, and you'll lose your PDF.

Refining print layout

You will doubtless want to refine your print layout by editing

your markdown and adding custom CSS to your print `.scss` file.

For instance, to save widows and orphans, when using the Classic theme, you can tighten letter-spacing by adding `{:.tighten}` and `{:.loosen}` tags to the lines after paragraphs.

And to add manual hyphens to improve spacing, use discretionary hyphens (aka soft hyphens) by adding the HTML entity `&shy;` where you want the soft hyphen.

Faster print refinement

When refining print layout (e.g. fixing widows and orphans) with the EBWorkflow, this is happening:

1. You edit the markdown
2. Jekyll generates new HTML
3. You convert to PDF in Prince
4. You open the new PDF to see your changes.

There are two delays in this process:

1. Jekyll can take several seconds to regenerate the entire book (even with `--incremental` enabled)
2. You have to click 'Convert' and Prince must regenerate the file(s)
3. If your PDF viewer locks the file you're viewing, you have to close it before generating a new PDF in Prince.

To speed this up:

1. Temporarily stop Jekyll generating any files you aren't working on. In `_config.yml`, add a line that excludes those

files. In this example, I've listed the CSS folder and all my book's chapters, and them commented out with `#` the one I'm working on, so that Jekyll does regenerate that:

```
# Jekyll will ignore these files and folders.
# Useful for temporarily speeding up processing for
particular files.
exclude: [
css,
book-one/0-0-cover.md,
book-one/0-1-half-title.md,
book-one/0-2-titlepage.md,
book-one/0-3-copyright.md,
book-one/0-4-contents.md,
#book-one/1.md,
book-one/2.md,
book-one/3.md,
]
```

2. Using Prince there isn't a universal way to speed up conversion, except to only convert the files you really need to be working on. (If you've used Javascript in your HTML, that can slow Prince down. We wrap scripts, like Google Analytics tags, in a test that checks for Prince.)

3. Use a PDF Viewer that doesn't lock the file.
    - On Windows, Sumatra is perfect for this, unlike Acrobat and Windows Preview, which locks the file. If you edit/regenerate a PDF it has open, Sumatra will allow that to happen and will automatically reload the new file, at the same page you had open.
    - On Mac OSX, Preview does the same, though we've found it doesn't open the regenerated file to the page you were on, sending you back to the start of the book, which is a pain. A much better alternative on OSX is Skim, an open-source PDF reader. In its Preferences > Sync, set Skim to watch for file changes and update automatically. You may also want to set PDFs to open to the last viewed page in Preferences > General.

Managing hyphenation in Prince

Our default stylesheets ask Prince to hyphenate paragraphs and lists (`p, ul, ol, dl`), with a few exceptions (such as text on the title and contents pages). Prince includes a range of hyphenation dictionaries by default, which do a good job. However, you might need to add dictionaries or lists of specific words that Prince doesn't support. You can find `.dic` files online for various languages and specialities, or you can compile your own.

We provide a blank `.dic` file for you to add your own list of hyphenation rules to. It is at `css/dictionaries/hyph.dic`. Our `print.css` asks Prince to look here when hyphenating.

A `.dic` file is a plain-text file with one word or word-fragment on each line. Each one is called a pattern.

- If the pattern starts with a `.`, it will apply to, or match, any word that starts with that pattern. E.g. `.foo` will match the words 'food' and 'foobar' but not 'fastfood'.
- If the pattern ends with a `.`, it will apply to any word that ends with that pattern. E.g. `port.` will match 'port' and 'sport' but not 'portico'.
- Thus, if the pattern starts *and* ends with a `.`, it will apply to only that pattern exactly. E.g. `.port.` will only ever match 'port'.

To show where a word or word-fragment can hyphenate, you add digits (1 to 9) to the pattern. The digits have special meanings:

- Insert odd digits (1, 3, 5, 7, 9) where the word may hyphenate.
- Insert even digits (2, 4, 6, 8) where the word should not hyphenate.
- The higher the number, the more important the rule. That is, a `1` says 'hyphenate here if you must', but a `9` says 'this is the best place to hyphenate'. A 2 says 'don't hyphenate here if you can help it', but an `8` says 'Do not, not, not hyphenate here.'

For user discussion, see the Prince forums here: If you need to hack Prince's built-in hyphenation dictionaries more deeply, see this forum post.

Epub output

Assembling the epub

We like to assemble our epubs (as EPUB2 for compatibility with older ereaders) in Sigil. If we're not changing something major, it takes five minutes. (See the pro tip below for an even quicker way.)

1. Put the HTML files from `_site` into your `Text` folder.
2. Put the finished `epub.css` (from `_site/css`) into your Sigil

`Styles` folder. (The `epub.css` file is a trimmed version of `screen.css`. It does not link to font files and avoids CSS3 features, like @fontface, some pseudo classes and media queries, to work better with popular readers with poor or buggy CSS support, such as Adobe Digital Editions.)

3. If your book has a child stylesheet, add that, too.
4. Replace any SVG images in the `Images` folder with JPG equivalents. And:
5. Search-and-replace any links to .svg in your HTML files with .jpg.
6. Replace the links to `screen.css` in your `<head>` elements with links to `epub.css`.
7. Remove the link to the print CSS in your `<head>` elements.
8. Copy any fonts into the `Fonts` folder, if you want them embedded. (If you don't want to embed fonts, remove any `@font-face` rules from your stylesheet to avoid file-not-found validation errors. We don't recommend embedding fonts unless they are required for meaning or unusual character sets.)
9. Search-and-replace to remove the `nav-bar` div (the link to `/` won't validate in an epub because it's not reachable). To find the nav-bar div in Sigil, use this DotAll Regex search:

```
(?s).<div class="non-printing"
id="nav-bar">(.*)<!--#nav-bar-->
```

   (You may need to reverse the order of the class and id attributes.)
10. Search-and-replace to remove the `footer` div (it's unnecessary in an ebook, and its links may break anyway). To

remove the footer div in Sigil, use this DotAll Regex search and replace with nothing:

```
(?s).<div class="non-printing"
id="footer">(.*)<!--#footer-->
```

11. Remove videos in iframes (iframes are invalid in EPUB2 XHTML 1.1). We recommend replacing videos with a link to an online version, e.g. to a YouTube page. This is best done manually. Search for `videowrapper` to find instances of embedded videos. The DotAll regex for finding the video wrapper is:

```
(?s).<div class="videowrapper
non-printing">(.*)</div><!--.videowrapper-->
```

To **replace** the standard video wrapper with a link to the video:

- Search (with DotAll regex) for:

```
(?s).<div class="videowrapper
non-printing">(.*)src="(.*?)"(.*)</div>(.*?)<!--.videowrapper-->
```

This will find the videowrapper and store the URL of the embedded video in memory.

- Replace with

```
<a href="\2" class="button">Watch</a>
```

This will replace the entire wrapper with a link to the same iframe URL it memorised (at `\2`). Replace `Watch` with whatever phrase you want to be the clickable text.

12. If your book includes endnotes (kramdown footnotes), replace `fnref:` with `fnref-` and `fn:` with `fn-`. (Background: If you have a colon in any element ID – for instance if you've used kramdown's footnote syntax – EpubCheck will return an 'invalid NCName' error. You need to replace those colons with another character. If your invalid IDs follow a set pattern (as kramdown's footnote references do), you can replace-all quickly.)
13. Add basic metadata to your epub using Sigil's Metadata Editor. Include at least:
    - title: subtitle
    - author
    - date of creation
    - publisher
    - ISBN (or other identifier like a UUID)
    - Relation ISBN (if any; we use the print ISBN as a parent ISBN)
14. Add semantics (right click the file name in Sigil for the semantics context menu) to:
    - key HTML files
    - the cover JPG.
15. Generate the epub's table of contents (Tools > Table Of Contents…). This TOC is generated only from the headings (`h1` to `h6`) in the text. So it does not include the cover, which has no heading, or any other files without a heading (e.g. sometimes the copyright page). If you need to add a cover to the TOC:
    a. Go to Tools > Table Of Contents > Edit Table Of Contents…
    b. Click on the first entry in the TOC list.

c. Click 'Add Above'.
d. Click 'Select Target' and select the cover HTML file (usually `0-0-cover.html`).
e. In the blank space under 'TOC Entry', double-click and type 'Cover'.
f. Click Okay.
g. Use the same process for adding any other files you need to add to the TOC.

16. If fonts are important (you've either embedded fonts or the difference between serif and sans-serif is semantically significant), add iBooks XML. (See below for detail.)
17. Check that the cover works, using your own cover-image jpg. For info on improving your epub cover layout, see the `cover.xhtml` and cover CSS snippets on our Knowledge Base. (We've already added the relevant cover CSS snippets to `epub.css`.)
18. Validate the epub in Sigil and fix any validation errors. Sigil let's some things past that EpubCheck flags, so also validate with EpubCheck directly. You can use:
    - the IDPF's online version of EpubCheck
    - epubcheck installed locally, and run from the command line; or
    - pagina EPUB-Checker.

For general guidance on creating epubs with Sigil, check out EBW's training material and the Sigil user guide.

Quicker epub output

Our template includes a Jekyll layout specifically for creating epubs. To use it, change the default `layout` in `_config.yml` (globally or for a given book's folder path) to `epub`. We include both options, and comment one of them out. So just switch which one is commented out:

```
layout: "epub"
#layout: "default"
```

Remember to restart Jekyll after changing `_config.yml` for changes to take effect, and change it back to `default` afterwards (`default` is necessary for print and web output).

The `epub` layout lets you skip a few of the steps listed above, because it:

- links to the epub CSS you specify in `_config.yml` (using Sigil's standard `../Styles/` path)
- omits the nav bar and footer (no need to search and replace these)
- includes required epub metadata (if you've included it in your `_config.yml` file).

To get the metadata to import to Sigil, you must *open* one of your book's HTML files in Sigil (the cover is best, since it's the first file). That is, don't 'Add Existing Files…' to a new, blank epub. Only by opening a single HTML file (as in 'File > Open…', then select the HTML file) will Sigil read and import the file's Dublin Core metadata. After that, you can add the remaining files in Sigil using 'Add Existing Files…'.

Quick-epub process checklist

If you've generated HTML using the `layout: epub` setting in `_config.yml`, your basic process in Sigil is as follows.

1. File > Open… and select the first HTML file in `_site/yourbook`.
2. Right-click the Text folder > Add Existing Files… and select the remaining HTML files for the epub.
3. Right-click the Text or Styles folder > Add Existing Files… and select the epub's CSS file in `_site/css`.
4. Tools > Table Of Contents > Generate Table Of Contents
5. Add file semantics (right-click HTML files > Add Semantics… and right-click the cover jpg > Cover Image).
6. Save, and validate with the Flightcrew plugin and separately with EPUBCheck.

Depending on your needs, you may also need to:

- search-and-replace (as described above) for SVG images, footnotes, or video;
- add the cover (or other files) to your table of contents (as described above)
- add font files if you're embedding fonts;
- split large files (as described below)
- add the iBooks display-options XML (as described below).

Splitting large files

If you have very large text files that, in the epub output, you'd like to split up into separate HTML files, Sigil can help. Using this tag in HTML, you can mark where Sigil must split your HTML

file(s):

```
<hr class="sigil_split_marker" />
```

To create that in kramdown:

```
***
{:.sigil_split_marker}
```

Also, remember to hide those markers in print output (and web and elsewhere as needed) with this in your CSS:

```
.sigil_split_marker {
        display: none;
}
```

Then, when you're assembling the epub in Sigil, just run Edit > Split At Markers.

Sigil will then split the HTML file into separate HTML files at the markers, and remove the `<hr>` element.

A common use case for this is books with end-of-book endnotes. To create end-of-book endnotes using kramdown footnotes you must put all content with endnotes in one markdown (and therefore HTML) file. This file is too large for sensible epub use, so splitting is important. Sigil is smart enough to update your internal links when you run 'Split At Markers'.

> **NB: Before running Split At Markers: save, close, and reopen your epub.** At least till Sigil 0.9.3, there is an issue with updating internal links when using Split At Markers. In order for internal links to update correctly, Sigil must

first have rewritten all link paths to HTML files according to its `../Text/` folder structure (e.g. the links to chapters in a Table of Contents file). Sigil only rewrites all these paths when an epub file is opened. So to make sure links are udpated when running Split At Markers, you need to save, close, and reopen the epub first. This may be fixed from Sigil 0.9.5.

Mobi conversion

If you need a MOBI file for Kindle, we recommend putting your EPUB into Kindlegen. If you don't want to use the command-line, just open the EPUB with the Kindle Previewer, which automatically converts to MOBI using Kindlegen and saves the MOBI file to a folder beside your EPUB.

If Kindlegen cannot convert the EPUB, we've found that adding it to Calibre first, then (without converting) give Calibre's version to Kindlegen.

Calibre gives you greater control over specific ebook conversions, but we've found Kindlegen converts some CSS better (e.g. floats and borders).

If you need to dig into a mobi file's code to troubleshoot, try the KindleUnpack plugin for Calibre.

Adding iBooks display-options file

If you need to add the `com.apple.ibooks.display-options.xml` file to your epub for iBooks display options, you can use the Add-iBooksXML plugin in Sigil.

A very basic display-options file contains this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<display_options>
    <platform name="*">
        <option name="specified-fonts">true</option>
        <option name="interactive">false</option>
        <option name="fixed-layout">false</option>
        <option name="open-to-spread">false</option>
        <option name="orientation-lock">none</option>
    </platform>
</display_options>
```

The file should be in the epub's META-INF folder, which Sigil does not let you edit by default, hence the need for the plugin.

To install the plugin:

- Download the zip file.
- In Sigil, go to Plugins > Manage Plugins.
- Click Add Plugin and locate and select the zip file you downloaded.

To use the plugin:

- First, in a plain-text/code editor create an `com.apple.ibooks.display-options.xml` file containing only the XML shown above. If necessary, change the five options settings in it. Save the XML file with your source material for the book for future use/reference.
- In Sigil, with the epub open, go to Plugins > Edit > AddiBooksXML and find and select the `com.apple.ibooks.display-options.xml` file you just created.

Troubleshooting and tips

Markdown

Use a good text editor that colour-codes markdown as you work. We like Sublime Text (with MarkdownEditing installed and set to MultiMarkdown) or Notepad++ (with this markdown highlighter).

- If you're working on Windows, set your default character encoding for your documents to 'UTF-8 without BOM', aka UTF-8, and *not* a 'UTF-with-BOM' option. (Jekyll will break

if you don't.)

- To check how your markdown converts to HTML while you work, you can use this Online Kramdown Editor by Daniel Perez Alvarez.
- Keep the kramdown quick reference handy.
- In lists, kramdown lets you use a space *or* a tab between the list marker (e.g. `*` or `1.` etc.) and the list text. If only to solve an issue with nesting blockquotes in lists, *use a tab* between the list marker and the start of the list text, and the same tab at the start of the blockquote line. That is, the indentation (the tab) must be exactly the same for the blockquote to nest correctly in the list. (A local Jekyll instance may correctly parse nested lists even with a space after the list marker and a tab before the blockquote `>`. But GitHub Pages is much stricter and requires exactly the same indentation.) E.g. see our book Newborn Care 12-5.
- Keep spans within block elements. For instance, if you have two paragraphs in italic, don't start the italics with `*` in the first paragraph and end the span with a second `*` in the next paragraph. The HTML needs one span (e.g. `<em>` span) in the first para, and another in the second para. The converter isn't smart enough to split your intended italics into two spans. Rather end the first span in the first para, and start another one in the second.

Jekyll and GitHub

- When running Jekyll locally, and *if* your repo is a project using GitHub Pages (not an organisation or user site), you'll need to add `--baseurl ''` when running Jekyll at the command line. Here's how and why.
- You may get different results between a local Jekyll install and GitHub Pages, even if both are using kramdown. Always check (at least spot check) both places.
- Do not use a colon `:` in the title you include in your YAML header (inside the `---`s at the tops of files). For instance, you can't have `title: Beans: The musical fruit`. Jekyll will break, unsure if you're trying to map a second value to the YAML key. You can either use another character (e.g. `title: Beans—The musical fruit`) or use the entity `&#58;` for the colon (`title: Beans&#58; The musical fruit`).
- We recommend setting `.gitignore` to ignore the `_site` folder, where Jekyll will store HTML output locally. If you choose *not* to `.gitignore` your `_site` folder, it'll contain (and sync to GitHub) your local machine's most recent Jekyll HTML output. (The `_site` folder has nothing to do with what GitHub Pages publishes.) In theory, committing the `_site` folder makes it easy for collaborators without Jekyll to grab a book's output HTML from the repo. But it comes with problems: committers have a responsibility to make sure their Jekyll instance does a good job, and that their `_site` output is up-to-date with the latest changes to the underlying markdown. Importantly, if you have more than one committer on a book, you'll get lots of merge conflicts in the `_site` folder,

and this will make your head hurt. (We let `_site` sync to GitHub for this guide so that it includes our example output for reference.)

Typography examples

This chapter simply includes lots of examples of our workflow's default typography using the Classic theme.

This is a regular paragraph. If it follows a heading, list, definition list, blockquote or iframe, it'll be flush left (not indented). If you're looking at the HTML, ebook or print version, you should also check out the underlying markdown version, to see how to control styles with simple, plain-text formatting.

There is more detail in the workflow's README.

In the markdown source of this chapter, note the `chapter` style in the markdown version's YAML header – at the top of the document. That's important: it tells the stylesheet what part of the book this is.

Better than lorem ipsum

Who needs lorem ipsum when you can read the real thing? Here's some Cicero translated by E. S. Shuckburgh.

To Terentia, Tulliola, and Young Cicero (at Rome)

BRUNDISIUM, 29 APRIL—Yes, I do write to you less often than I might, because, though I am always wretched, yet when I write to you or read a letter from you, I am in such floods of tears that I cannot endure it. Oh, that I had clung less to life![1] I should at least never have known real sorrow, or not much of it, in my life. Yet if fortune has reserved for me any hope of recovering at any time any position again, I was not utterly wrong to do so: if these miseries are to be permanent, I only wish, my dear, to see you as soon as possible and to die in your arms, since neither gods,

To get all small caps, wrap the text in asterisks, tagged `{:.allsmallcaps}`

whom you have worshipped with such pure devotion, nor men, whom I have ever served, have made us any return.

> *This is a blockquote tagged as a* `{:.pull-quote}` *– that's useful.*

I only wish, my dear, to see you as soon as possible and to die in your arms.

I have been thirteen days at Brundisium in the house of M. Laenius Flaccus, a very excellent man, who has despised the risk to his fortunes and civil existence in comparison to keeping me safe, nor has been induced by the penalty of a most iniquitous law to refuse me the rights and good offices of hospitality and friendship.[2] May I sometime have the opportunity of repaying him! Feel gratitude I always shall. I set out from Brundisium on the 29th of April, and intend going through Macedonia to Cyzicus.

What a fall! What a disaster! What can I say? Should I ask you to come–a woman of weak health and broken spirit? Should I refrain from asking you? Am I to be without you, then?

But what is to become of my darling Tullia? You must see to that now: I can think of nothing. But certainly, however things turn out, we must do everything to promote that poor little girl's married happiness and reputation. Again, what is my boy Cicero to do? Let him, at any rate, be ever in my bosom and in my arms.

> This is a regular blockquote. I can't write more. A fit of weeping hinders me. I don't know how you have got on; whether you are left in possession of anything, or have been, as I fear, entirely plundered. Piso, as you say, I hope will always be our friend.

As to the manumission of the slaves you need not be uneasy. To begin with, the promise made to yours was that you would treat them according as each severally deserved. So far Orpheus has behaved well, besides him no one very markedly so. With the rest of the slaves the arrangement is that, if my property is forfeited, they should become my freedmen, supposing them to be able to maintain at law that status. But if my property remained in my ownership, they were to continue slaves, with the exception of a very few. But these are trifles.

To return to your advice, that I should keep up my courage and not give up hope of recovering my position, I only wish that there were any good grounds for entertaining such a hope. As it

> This is a blockquote tagged as a `{:.box}` and placed at the bottom of the page (in print output) with `.float-bottom`.
>
> I think the best course is this: if there is any hope of my restoration, stay to promote it and push the thing on.
>
> But if, as I fear, it proves hopeless, pray come to me by any means in your power. Be sure of this, that if I have you I shall not think myself wholly lost.

is, when, alas! shall I get a letter from you? Who will bring it me? I would have waited for it at Brundisium, but the sailors would not allow it, being unwilling to lose a favourable wind.

For the rest, put as dignified a face on the matter as you can, my dear Terentia. Our life is over: we have had our day: it is not any fault of ours that has ruined us, but our virtue. I have made no false step, except in not losing my life when I lost my honours. But since our children preferred my living, let us bear everything else, however intolerable. And yet I, who encourage you, cannot encourage myself.

If you want to start a new section of paragraphs, with space above and no indent, tag the paragraph with `{:.first}` in the markdown, like this one. I have sent that faithful fellow Clodius Philhetaerus home, because he was hampered with weakness of the eyes. Sallustius seems likely to outdo everybody in his attentions. Pescennius is exceedingly kind to me; and I have hopes that he will always be attentive to you. Sicca had said that he would accompany me; but he has left Brundisium.

Take the greatest care of your health, and believe me that I am more affected by your distress than my own. My dear Terentia, most faithful and best of wives, and my darling little daughter, and that last hope of my race, Cicero, good-bye!

A valediction, 29 April, from Brundisium

Still better than lorem ipsum

Here's some more Cicero translated by E. S. Shuckburgh.

To his Brother Quintus (in Sardinia)

ROME, 12 FEBRUARY—I have already told you the earlier proceedings; now let me describe what was done afterwards:

- This is a list with a nested sublist.
- The legations were postponed from the 1st of February to the 13th.
- On the former day our business was not brought to a settlement.
- On the 2nd of February Milo appeared for trial.
    - Pompey came to support him.
    - Marcellus spoke on being called upon by me.
    - We came off with flying colours.
- The case was adjourned to the 7th.

Meanwhile (in the senate), the legations having been postponed to the 13th, the business of allotting the quaestors and furnishing the outfit of the praetors was brought before the house. But nothing was done, because many speeches were interposed denouncing the state of the Republic.

Gaius Cato published his bill for the recall of Lentulus, whose son thereon put on mourning. On the 7th Milo appeared. Pompey

This is a sidenote in its own paragraph. It floats to the right of the para-

spoke, or rather wished to speak. For as soon as he got up Clodius's ruffians raised a shout, and throughout his whole speech he was interrupted, not only by hostile cries, but by personal abuse and insulting remarks. However, when he had finished his speech—for he shewed great courage in these circumstances, he was not cowed, he said all he had to say, and at times had by his commanding presence even secured silence for his words–well, when he had finished, up got Clodius. Our party received him with such a shout—for they had determined to pay him out–that he lost all presence of mind, power of speech, or control over his countenance.

This went on up to two o'clock – Pompey having finished his speech at noon – and every kind of abuse, and finally epigrams of the most outspoken indecency were uttered against Clodius and Clodia. Mad and livid with rage Clodius, in the very midst of the shouting, kept putting questions to his claque: "Who was it who was starving the commons to death?" His ruffians answered, "Pompey." "Who wanted to be sent to Alexandria ?" They answered, "Pompey." "Who did they wish to go ?" They answered, "Crassus." The latter was present at the time with no friendly feelings to Milo.

About three o clock, as though

Here's another sidenote, but applied to a blockquote. Blockquotes as sidenotes are best if your content might be read with default browser CSS, because it keeps them distinct from the paragraph flow.

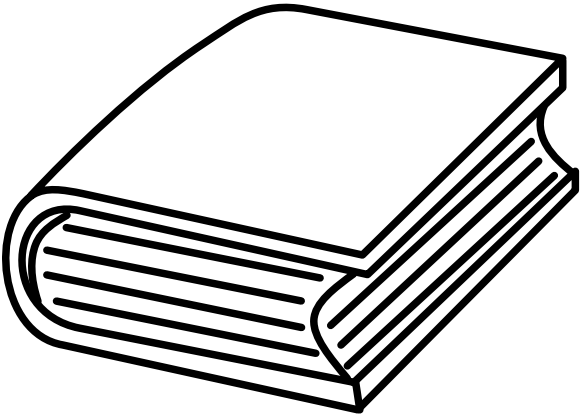
This is a sidenote using an inline span. It's exactly aligned with its position in the source text. On screen, a browser must support CSS to float the foot-

at a given signal, the Clodians began spitting at our men.

1. A numbered list. Kramdown will create a correctly numbered list in HTML even if the list numbers in the markdown aren't in sequence.
2. There was an outburst of rage.
3. They began a movement for forcing us from our ground.
4. Our men charged: his ruffians turned tail.
5. Clodius was pushed off the rostra: and then we too made our escape for fear of mischief in the riot.
6. The senate was summoned into the Curia: Pompey went home.

However, I did not myself enter the senate-house, lest I should be obliged either to refrain from speaking on matters of such gravity, or in defending Pompey (for he was being attacked by Bibulus, Curio, Favonius, and Servilius the younger) should give offence to the loyalists.
The business was adjourned to the next day. Clodius fixed the Quirinalia (17th of February) for his prosecution. On the 8th the senate met in the temple of Apollo, that Pompey might attend. Pompey made an impressive speech.

> That day nothing was concluded. On the 9th in the temple of Apollo a degree passed the senate "that what had taken place on the 7th of February was treasonable."
>
> On this day Cato warmly inveighed against Pompey, and throughout his speech arraigned him as though he were at the bar. He said a great deal about me, to my disgust, though it was in very laudatory terms.



This is a figure caption. Formatting works if *the image and caption are two consecutive paragraphs in a blockquote.*

When he attacked Pompey's perfidy to me, he was listened to in profound silence on the part of my enemies. Pompey answered him boldly with a palpable allusion to Crassus, and said outright that "he would take better precautions to protect his life than Africanus had done, whom C. Carbo had assassinated."

Accordingly, important events appear to me to be in the wind. For Pompey understands what is going on, and imparts to me

- that plots are being formed against his life,
- that Gaius Cato is being supported by Crassus,
- that money is being supplied to Clodius,
- that both are backed by Crassus and Curio, as well as by Bibulus and his other detractors,
- that he must take extraordinary precautions to prevent being

> overpowered by that demagogue–with a people all but wholly alienated, a nobility hostile, a senate ill-affected, and the younger men corrupt.

So he is making his preparations and summoning men from the country.

On his part, Clodius is rallying his gangs: a body of men is being got together for the Quirinalia. For that occasion we are considerably in a majority, owing to the forces brought up by Pompey himself: and a large contingent is expected from Picenum and Gallia, to enable us to throw out Cato's bills also about Milo and Lentulus.

On the 10th of February an indictment was lodged against Sestius for bribery by the informer Cn. Nerius, of the Pupinian tribe, arid on the same day by a certain M. Tullius for riot. He was ill. I went at once, as I was bound to do. to his house, and put myself wholly at his service: and that was more than people expected, who thought that I had good cause for being angry with him.

A paragraph tagged as a `{:.box}`. The result is that my extreme kindness and grateful disposition are made manifest both to Sestius himself and to all the world, and I

This is a note that on screen will appear as a sidenote, and in print will appear at the bottom of the page. To do this, add `.bottom` to the `.sidenote` tag: `{:.sidenote .bottom}`.

shall be as good as my word. But this same informer Nerius also named Cn. Lentulus Vatia and C. Cornelius to the commissioners.

On the same day a decree passed the senate "that political clubs and associations should be broken up, and that a law in regard to them should be brought in, enacting that those who did not break off from them should be liable to the same penalty as those convicted of riot."

On the 10th of February I spoke in defence of Bestia on a charge of bribery before the praetor Cn. Domitius, in the middle of the forum and in a very crowded court; and in the course of my speech I came to the incident of Sestius, after receiving many wounds in the temple of Castor, having been preserved by the aid of Bestia. Here I took occasion to pave the way beforehand for a refutation of the charges which are being got up against Sestius, and I passed a well-deserved encomium upon him with the cordial approval of everybody. He was himself very much delighted with it. I tell you this because you have often advised me in your letters to retain the friendship of Sestius.

Sestius

This is an example of definition-list typography. Publius Sestius was a Roman senator in the 1st century BC. He was a praetor in 53 BC, as well as a friend and ally of Cicero, by whom he was defended in 56 BC. Upon the outbreak of Caesar's Civil War he joined the party of Pompey, having become

the governor of Cilicia. According to Plutarch's Life of Brutus he was accompanied by Marcus Brutus to his province, but Sestius subsequently went over to Caesar, who sent him into Cappadocia in 48 BC.

This is I am writing this on the 12th of February before daybreak; the day on which I am to dine with Pomponius on the occasion of his wedding.

Our position in other respects is such as you used to cheer my despondency by telling me it would be–one of great dignity and popularity: this is a return to old times for you and me effected, my brother, by your patience, high character, loyalty, and, I may also add, your conciliatory manners. The house of Licinius, near the grove of Piso, has been taken for you. But, as I hope, in a few months time, after the 1st of July, you will move into your own.

Note

You can tag almost any element with `{:.box}`, including definition lists like this.

Some excellent tenants, the Lamiae, have taken your house in Carinie. I have received no letter from you since the one dated Olbia. I am anxious to hear how you are and what you find to amuse you, but above all to see you yourself as soon as possible.

Take care of your health, my dear brother, and though it is winter time, yet reflect that after all it is Sardinia that you are in.

13 February

Poetry

Best-practice poetry layout indents a poem till its longest line is centred on the page. To achieve this, we put the entire poem, including the title, in a 'verse' blockquote and specify how wide this poem should be on the page.

To One Who Has Been Long in City Pent

To one who has been long in city pent,
'Tis very sweet to look into the fair
And open face of heaven,—to breathe a prayer
Full in the smile of the blue firmament.
Who is more happy, when, with heart's content,
Fatigued he sinks into some pleasant lair
Of wavy grass, and reads a debonair
And gentle tale of love and languishment?

Returning home at evening, with an ear
Catching the notes of Philomel,—an eye

Watching the sailing cloudlet's bright career,
He mourns that day so soon has glided by:
E'en like the passage of an angel's tear
That falls through the clear ether silently.

Here's a long poetry example.

Gerontion

Thou hast nor youth nor age
But as it were an after dinner sleep
Dreaming of both.

Here I am, an old man in a dry month,
Being read to by a boy, waiting for rain.
I was neither at the hot gates
Nor fought in the warm rain
Nor knee deep in the salt marsh, heaving a cutlass,
Bitten by flies, fought.
My house is a decayed house,
And the jew squats on the window sill, the owner,
Spawned in some estaminet of Antwerp,
Blistered in Brussels, patched and peeled in London.
The goat coughs at night in the field overhead;
Rocks, moss, stonecrop, iron, merds.
The woman keeps the kitchen, makes tea,
Sneezes at evening, poking the peevish gutter.

I an old man,

A dull head among windy spaces.

Signs are taken for wonders. "We would see a sign":
The word within a word, unable to speak a word,
Swaddled with darkness. In the juvescence of the year
Came Christ the tiger

In depraved May, dogwood and chestnut, flowering Judas,
To be eaten, to be divided, to be drunk
Among whispers; by Mr. Silvero
With caressing hands, at Limoges
Who walked all night in the next room;
By Hakagawa, bowing among the Titians;
By Madame de Tornquist, in the dark room
Shifting the candles; Fraulein von Kulp
Who turned in the hall, one hand on the door. Vacant shuttles
Weave the wind. I have no ghosts,
An old man in a draughty house
Under a windy knob.

After such knowledge, what forgiveness? Think now
History has many cunning passages, contrived corridors
And issues, deceives with whispering ambitions,
Guides us by vanities. Think now
She gives when our attention is distracted
And what she gives, gives with such supple confusions
That the giving famishes the craving. Gives too late
What's not believed in, or if still believed,
In memory only, reconsidered passion. Gives too soon
Into weak hands, what's thought can be dispensed with

Till the refusal propagates a fear. Think
Neither fear nor courage saves us. Unnatural vices
Are fathered by our heroism. Virtues
Are forced upon us by our impudent crimes.
These tears are shaken from the wrath-bearing tree.

The tiger springs in the new year. Us he devours. Think at last
We have not reached conclusion, when I
Stiffen in a rented house. Think at last
I have not made this show purposelessly
And it is not by any concitation
Of the backward devils.
I would meet you upon this honestly.
I that was near your heart was removed therefrom
To lose beauty in terror, terror in inquisition.
I have lost my passion: why should I need to keep it
Since what is kept must be adulterated?
I have lost my sight, smell, hearing, taste and touch:
How should I use it for your closer contact?

These with a thousand small deliberations
Protract the profit of their chilled delirium,
Excite the membrane, when the sense has cooled,
With pungent sauces, multiply variety
In a wilderness of mirrors. What will the spider do,
Suspend its operations, will the weevil
Delay? De Bailhache, Fresca, Mrs. Cammel, whirled
Beyond the circuit of the shuddering Bear
In fractured atoms. Gull against the wind, in the windy straits
Of Belle Isle, or running on the Horn,

White feathers in the snow, the Gulf claims,
And an old man driven by the Trades
To a sleepy corner.

Tenants of the house,
Thoughts of a dry brain in a dry season.

On screen, there's a button here. This one won't show in print output because it's tagged `.non-printing`. To create a button, just add `{:.button}` directly after a link, like this: `[Go to Contents](0-3-contents.html){:.button}`.

1. This is an endnote. Notes created like this follow kramdown's footnote syntax (kramdown uses the term footnote to refer to endnotes, unfortunately).
2. This is another endnote. Note how kramdown automatically numbers the these correctly, even when the markdown doesn't. The reference is designed to appear superscript. We can also do subscript in markdown, like H_2o, but we have to use HTML `<sub>` tags. Our workflow doesn't yet handle complex maths by default, but it may in future, using MathJax.