



softcover

a manual, guide, and reference

Michael Hartl

The Softcover Book

Frictionless self-publishing

Michael Hartl

Contents

| | | |
|----------|---|-----------|
| 1 | Getting started | 1 |
| 1.1 | The Softcover typesetting system | 3 |
| 1.1.1 | Installing Softcover | 5 |
| 1.1.2 | Creating a Softcover book | 9 |
| 1.1.3 | HTML and the Softcover server | 12 |
| 1.1.4 | Building ebooks | 17 |
| 1.1.5 | Cover images | 24 |
| 1.2 | Publishing to the Softcover website | 25 |
| 1.2.1 | Publishing ebooks | 25 |
| 1.2.2 | One command to rule them all | 26 |
| 1.2.3 | Articles | 27 |
| | | |
| 2 | Introduction to Markdown | 29 |
| 2.1 | Headings | 30 |
| 2.2 | Text formatting | 30 |
| 2.2.1 | Blockquotes | 31 |
| 2.2.2 | Source code | 32 |
| 2.3 | Links and images | 32 |
| 2.3.1 | PDF/PNG images | 34 |
| 2.3.2 | Screenshots and other large images | 36 |
| 2.4 | Lists | 37 |
| 2.4.1 | Numbered lists | 37 |
| 2.4.2 | Unnumbered lists | 39 |
| 2.4.3 | Paragraphs in lists | 40 |

| | | |
|----------|---|-----------|
| 3 | Softcover-flavored Markdown | 43 |
| 3.1 | The kramdown extensions | 46 |
| 3.1.1 | Tables | 47 |
| 3.1.2 | Numbered footnotes | 48 |
| 3.1.3 | Miscellaneous features | 49 |
| 3.2 | Other advanced enhancements | 50 |
| 3.2.1 | GitHub-flavored fenced code blocks | 50 |
| 3.2.2 | Leanpub-style language blocks | 52 |
| 3.2.3 | Code inclusion | 54 |
| 3.2.4 | Embedded math | 55 |
| 3.3 | Embedded \LaTeX | 56 |
| 3.3.1 | \LaTeX commands | 56 |
| 3.3.2 | Labels and cross-references | 59 |
| 3.3.3 | Tabular and tables | 61 |
| 3.3.4 | Figures | 64 |
| 3.3.5 | Code listings | 67 |
| 3.3.6 | Aside boxes | 68 |
| 3.3.7 | Math and numbered equations | 69 |
| 3.3.8 | Colored text | 71 |
| 3.3.9 | Inputting contents of other files | 72 |
| 4 | Customization and advanced options | 75 |
| 4.1 | Command-line interface | 75 |
| 4.1.1 | Customizing builds | 75 |
| 4.1.2 | Customizing deploys | 76 |
| 4.2 | Commands and styles | 77 |
| 4.2.1 | Custom commands | 77 |
| 4.2.2 | HTML style | 78 |
| 4.2.3 | EPUB/MOBI style | 79 |
| 4.2.4 | PDF style | 80 |
| 4.2.5 | Advanced figure placement | 82 |
| 4.3 | Foreign-language support | 84 |
| 4.3.1 | Polyglossia and <code>lang.yml</code> | 85 |
| 4.3.2 | Terrifyingly advanced comments on Hungarian | 87 |

| | | |
|----------|---|------------|
| 4.4 | Detailed refinements | 88 |
| 4.4.1 | Overfull hboxes | 88 |
| 4.4.2 | Problems with labels and cross-references | 89 |
| 4.4.3 | EPUB validation | 90 |
| 5 | Marketing and selling | 93 |
| 5.1 | Publishing a book | 93 |
| 5.2 | Screencasts and other media | 94 |
| 5.3 | Book description | 95 |
| 5.4 | Marketing page | 97 |
| 5.4.1 | Prices | 97 |
| 5.4.2 | Author information and testimonials | 99 |
| 5.4.3 | Frequently Asked Questions | 101 |
| 5.4.4 | Additional information | 102 |
| 5.5 | Site settings and customizations | 102 |
| 5.5.1 | Access options | 103 |
| 5.5.2 | Custom domains | 105 |
| 5.5.3 | Google Analytics | 105 |
| 5.5.4 | Miscellaneous content | 106 |
| 5.6 | A typical launch sequence | 107 |
| 6 | Poly$\text{T}_{\text{E}}\text{X}$ tutorial | 109 |
| 6.1 | Poly $\text{T}_{\text{E}}\text{X}$ basics | 109 |
| 6.1.1 | Generating a Poly $\text{T}_{\text{E}}\text{X}$ book | 109 |
| 6.1.2 | From Markdown to Poly $\text{T}_{\text{E}}\text{X}$ | 110 |
| 6.1.3 | Poly $\text{T}_{\text{E}}\text{X}$ vs. $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ | 110 |
| 6.2 | Included commands | 111 |
| 6.2.1 | Colored text | 111 |
| 6.2.2 | Code inclusion | 112 |

Chapter 1

Getting started

This is *The Softcover Book*—the manual for *Softcover*, a publishing platform for technical authors. Softcover consists of two main parts: a state-of-the-art [open-source ebook typesetting system](#) (Section 1.1), and an [online platform](#) for publishing, marketing, and selling ebooks and other digital goods (Section 1.2). Based on the technology used to make the *Ruby on Rails Tutorial* book and *The Tau Manifesto*, Softcover makes publishing *frictionless* by allowing authors to build and deploy ebooks and other digital goods with a single command.

The Softcover production toolchain and publishing platform are especially designed to help authors make the transition from “writing a book” to “building a business,” using the following [three-step plan](#):

1. Make ebooks, screencasts, etc.
2. Sell HTML and ebooks and multiple product bundles
3. Profit!!!

Of course, it’s not necessary to follow the Three Step Plan™ exactly, and Softcover can be used for many different purposes ([Box 1.1](#)).

Box 1.1. How to use Softcover

Softcover is a flexible tool, so it has many potential uses. The first use represents Softcover's principal design goal, but the others are valid possibilities as well:

- **Make an HTML book, bundle ebooks with other digital goods, and sell them from the [Softcover.io](https://softcover.io) online storefront**
- Charge for all products (including the HTML book) at [Softcover.io](https://softcover.io)
- Produce ebooks with the Softcover typesetting system and give them away
- Produce ebooks with Softcover and sell them using the [Softcover.io](https://softcover.io) online storefront
- Produce ebooks with Softcover and sell them from your own website
- Use Softcover to make ebooks out of technical documentation and host them on an internal website

Softcover is inspired by the philosophy of *full author ownership*:

- **Own your content:** Authors retain copyright on all materials.
- **Own your production toolchain:** The Softcover typesetting system is open-source, so you aren't locked into a proprietary toolchain.
- **Own your traffic:** Softcover supports custom domains.
- **Own your customer list:** Authors get all relevant contact information and never have to use an intermediary to communicate with their customers.

In short, Softcover is a publishing platform I would be happy to use even if I weren't one of the founders of the company.

In the rest of this chapter, we'll cover the steps needed to install and use the Softcover typesetting system, which includes a command-line interface for building and publishing ebooks. Subsequent chapters cover Markdown, the default input format for Softcover ([Chapter 2](#) and [Chapter 3](#)), and Poly $\text{T}_{\text{E}}\text{X}$, a more complicated but more powerful input format based on the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ typesetting language ([Chapter 6](#)).

1.1 The Softcover typesetting system

In this section, we'll cover the basics of the Softcover open-source ebook typesetting system for technical authors. Its *raison d'être* is producing professional-grade multi-format ebooks from a common set of source files. In particular, Softcover accepts input in *Markdown*, a lightweight markup language, or *Poly $\text{T}_{\text{E}}\text{X}$* , a subset of the powerful $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ typesetting language, and outputs ebooks as HTML, EPUB, MOBI, and PDF. The Softcover system also comes with a local server that automatically rebuilds a book's HTML output when the source files change, so that your favorite text editor and web browser combine to form a real-time development environment for writing ebooks. Finally, authors can use Softcover to upload ebooks and other media files to the [Softcover website](#) with a single command ([Section 1.2](#)), thereby dramatically lowering the barrier to publishing, updating, and selling digital information products.

Originally developed under the name *Poly $\text{T}_{\text{E}}\text{X}nic$* ¹ to write the *Ruby on Rails Tutorial book* and *The Tau Manifesto*, Softcover has been rewritten and expanded as part of developing the [Softcover publishing platform](#). True to its origins, Softcover supports a wide range of features useful for writing technical books, including mathematical typesetting ([Eq. \(1.1\)](#))² and syntax-highlighted source code listings ([Listing 1.1](#)). It is also well-suited to writing non-technical books, with support for chapters, sections, cross-references, footnotes, lists, figures, tables, etc.—the only requirement is that the *author* be technical. (If you

¹Poly $\text{T}_{\text{E}}\text{X}nic$ is pronounced exactly like the English word *polytechnic*. The core input-to-output conversion is still handled by the `polytexnic` gem.

²[Eq. \(1.1\)](#) is written in *rationalized MKS units* (also known as “God’s units”), which set $\mu_0 = \epsilon_0 = 1$. (In addition to being beautiful, this choice of units gives us $c = 1/\sqrt{\mu_0\epsilon_0} = 1$ for free.)

know how to use the Unix [command line](#) and have a favorite [text editor](#), you are technical enough to use Softcover.)

$$\left. \begin{aligned} \nabla \cdot \mathbf{E} &= \rho \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} &= -\dot{\mathbf{B}} \\ \nabla \times \mathbf{B} &= \mathbf{J} + \dot{\mathbf{E}} \end{aligned} \right\} \text{Maxwell's equations} \quad (1.1)$$

Listing 1.1: “Maxwell’s equations of software” in Scheme.

```
;; Implements Lisp in Lisp.
;; Alan Kay called this feat "Maxwell's equations of software", because just as
;; Maxwell's equations contain all of electrodynamics, `eval` and `apply`
;; contain all of computing.

;; Evaluates an arbitrary Scheme S-expression.
(define (eval exp env)
  (cond ((self-evaluating? exp) exp)
        ((variable? exp) (lookup-variable-value exp env))
        ((quoted? exp) (text-of-quotation exp))
        ((assignment? exp) (eval-assignment exp env))
        ((definition? exp) (eval-definition exp env))
        ((if? exp) (eval-if exp env))
        ((lambda? exp)
         (make-procedure (lambda-parameters exp)
                          (lambda-body exp)
                          env))
        ((begin? exp)
         (eval-sequence (begin-actions exp) env))
        ((cond? exp) (eval (cond->if exp) env))
        ((application? exp)
         (apply (eval (operator exp) env)
                  (list-of-values (operands exp) env)))
        (else
         (error "Unknown expression type -- EVAL" exp))))

;; Applies a Scheme procedure to arbitrary arguments.
(define (apply procedure arguments)
  (cond ((primitive-procedure? procedure)
         (apply-primitive-procedure procedure arguments))
        ((compound-procedure? procedure)
         (eval-sequence
          (procedure-body procedure)
          (extend-environment
           (procedure-parameters procedure)
           arguments
           (procedure-environment procedure))))))
```

```
(else
  (error
    "Unknown procedure type -- APPLY" procedure))))
```

Naturally, *The Softcover Book* itself is written using Softcover. Indeed, you can consider this manual the *de facto* [spec](#) for the system: essentially everything that Softcover can do, this document does. Because the [source code of *The Softcover Book*](#) is available online, you can learn how to typeset anything you see in this manual simply by referring to the corresponding place in the source.

1.1.1 Installing Softcover

The Softcover system is open-source software, distributed as a Ruby gem under the permissive [MIT License](#). The `softcover` gem currently works with macOS and Linux, and we're looking for people to help us adapt it to other OSes. Join the [Softcover Google Group](#) to be part of that effort.

To get started with Softcover, first [install Ruby](#) (1.9.3 or higher) and [install RubyGems](#) if you don't have them already. Once you've done so, getting Softcover is usually a simple **gem install**:

```
$ gem install softcover
```

If you run into any permissions issues, either use **sudo**³ or (preferred) use [rbenv](#) to manage your Ruby environment and gems.

There have been some reports of installation issues on macOS (Mavericks and later), so if you run into trouble with the previous step, try this command instead:

```
$ gem install softcover -- --with-cppflags=-I/usr/local/opt/openssl/include
```

or possibly this:

³`sudo gem install <gem name>`

```
% gem install softcover -- \
    --with-cflags="-Wno-error=implicit-function-declaration"
```

or this:

```
% gem install softcover -- --with-cppflags=-I/usr/local/opt/openssl/include \
    --with-cflags="-Wno-error=implicit-function-declaration"
```

This installs the **softcover** command-line interface (CLI) for creating new books, building ebooks, and publishing ebooks and other digital assets to the [Softcover website](#). On some systems, you may have to install extra libraries; for example, on Ubuntu I needed to install **ruby1.9.1-dev** to get the `nokogiri` gem to install.

To build the full set of output formats, Softcover requires some external dependencies. The **softcover** command will prompt you to install each dependency at the appropriate time, but many users will find it more convenient to install all the dependencies at once. To check which dependencies need to be installed on your system, run **softcover check**:

```
$ softcover check
Checking Softcover dependencies...
Checking for LaTeX...      Found
Checking for ImageMagick... Found
Checking for Node.js...    Found
Checking for PhantomJS...  Found
Checking for Inkscape...   Found
Checking for Calibre...    Found
Checking for Java...       Found
Checking for EpubCheck...  Found
All dependencies satisfied.
```

In the unlikely case that you get the result above, congratulations—you have nothing to install. More likely, though, **softcover check** will indicate several missing dependencies. The output in this case includes URLs for all relevant software, but for convenience we also include them here:

- [LaTeX](#)

The \LaTeX download is *big*, so start downloading it now. Also, I strongly recommend installing a precompiled version of \LaTeX and *not* building it from source. Make sure to use the version of \LaTeX appropriate for your system from the link above; in particular, macOS users should use the [MacTeX package](#) rather than the version installed by Homebrew. Finally, several macOS users have reported having to restart their terminal program after installing MacTeX in order to enable the \LaTeX command-line programs (specifically, `xelatex`).

- [ImageMagick](#)

- [Node.js](#)

- [PhantomJS](#) (needed only for math output in EPUB and MOBI)

Development on the PhantomJS project has been suspended but the binaries should still work on most systems. If you have trouble installing PhantomJS on Linux, see [this Stack Overflow thread](#) to see if it helps. Finally, if you don't plan to use \LaTeX for mathematical typesetting in EPUB and MOBI output, you don't need PhantomJS at all.

- [Inkscape](#)

- [Calibre](#) with the command-line tools (built-in on Linux; on macOS, see below)

- [Java](#) (chances are you already have this one)

- [EpubCheck 4.0.1](#) (unzip and place in a directory on your path, i.e., `$HOME/bin`⁴)

On macOS, the Calibre command-line tools come included with calibre, but in order to make them available you have to put them on your PATH. Using a

⁴If `$HOME/bin` does not exist, you can create it using `mkdir $HOME/bin`. Then move EpubCheck there using `mv epubcheck-4.0.1 $HOME/bin`. Depending on your system, you might have to add `$HOME/bin` to the path by editing and sourcing `.bash_profile` (as shown in [Listing 1.2](#) and [Listing 1.3](#)).

text editor, put the contents of Listing 1.2 at the end of your `.bash_profile` file, and then run the `source` command in Listing 1.3 to update your shell.

Listing 1.2: Putting the Calibre executables on the PATH.

```
~/.bash_profile
.
.
.
export CALIBRE="/Applications/calibre.app/Contents/MacOS"
export PATH="$CALIBRE:$PATH"
```

Listing 1.3: Sourcing the bash profile.

```
$ source ~/.bash_profile
```

To see the commands supported by `softcover`, run `softcover help` at the command line, as shown in Listing 1.4.

Listing 1.4: Viewing available Softcover commands with `softcover help`.

```
$ softcover help
Commands:
softcover build, build:all           # Build all formats
softcover build:epub                 # Build EPUB
softcover build:html                 # Build HTML
softcover build:mobi                 # Build MOBI
softcover build:pdf                  # Build PDF
softcover build:preview              # Build book preview in all formats
softcover check                      # Check dependencies
softcover clean                      # Clean unneeded files
softcover config                    # View local config
softcover config:add key=value       # Add to your local config vars
softcover config:remove key         # Remove key from local config vars
softcover deploy                     # Build & publish book
softcover epub:validate, epub:check # Validate EPUB with epubcheck
softcover exercises                  # Add exercise id elements as spans
softcover help [COMMAND]            # Describe available commands...
softcover login                      # Log into Softcover account
softcover logout                     # Log out of Softcover account
softcover new <name>                # Generate new document directory structure
```



```
softcover open           # Open book on Softcover website
softcover publish        # Publish your book on Softcover
softcover publish:media  # Publish media
softcover server         # Run local server
softcover unpublish      # Remove book from Softcover
softcover version        # Return the version number (-v for short)
```

For convenience, Softcover also installs a shorter alias, **sc**, so you can (for example) get the current version number by typing any of the following:

```
$ softcover version
$ softcover -v
$ sc version
$ sc -v
```

1.1.2 Creating a Softcover book

We see from [Listing 1.4](#) that the way to generate a new Softcover book is with **softcover new <name>**.⁵ Let's try it out and see what happens; the results are shown in [Listing 1.5](#).

Listing 1.5: Generating an example book.

```
$ softcover new example_book
Generating directory: example_book
Creating chapters
Creating config
Creating epub
Creating epub/OEBPS
Creating epub/OEBPS/styles
Creating html
Creating html/jquery
Creating html/jquery/1.10.2
Creating html/stylesheets
Creating images
Creating images/figures
```

⁵Because Softcover books include a several elements that are specific to each book (including a symlink for the images directory, an internal book id, and a [UUID](#) included to fulfill a requirement of the EPUB standard), book directories should never be copied by hand to create new books. Instead, all new books should be generated using **softcover new**, and then any necessary files should be copied over individually.

```
Creating latex_styles
Creating screencasts
Creating .softcover-deploy
Creating Book.txt
Creating chapters/a_chapter.md
Creating chapters/another_chapter.md
Creating chapters/preface.md
Creating chapters/yet_another_chapter.md
Creating config/book.yml
Creating config/marketing.yml
.
.
.
Creating latex_styles/custom.sty
Creating latex_styles/custom_pdf.sty
Creating latex_styles/framed.sty
Creating latex_styles/softcover.sty
Creating latex_styles/upquote.sty
Creating README.md
Creating screencasts/.gitkeep
Done. Please update config/book.yml
```

Note: If your document is more naturally thought of as a single article rather than as a book, you should use the *article* format instead, as discussed in [Section 1.2.3](#). The tutorials at [Learn Enough to Be Dangerous](#) use this format.

The default book format generated by **softcover new** is *Markdown*—or, rather, a superset of Markdown that includes extensions to make it more suitable for writing longer documents. As discussed in [Chapter 3](#), this includes features such as select [kramdown](#) extensions (numbered footnotes, tables, etc.), GitHub-style code fencing, and embedded L^AT_EX. Authors who want more fine-grained control over their documents (or who already know L^AT_EX) can use **softcover new -p <name>** to generate a PolyT_EX template instead; see [Chapter 6](#) for details.

We see from [Listing 1.5](#) that **softcover new** generates a bunch of files, but the heart of it is the **chapters/** directory, which contains the Markdown source for the template book:

```
$ cd example_book
$ ls chapters/
a_chapter.md      preface.md
another_chapter.md yet_another_chapter.md
```

The names of the template form a progression (“A chapter”, “Another chapter”, “Yet *another* chapter”), but their order is not set by this progression. Rather, it is specified by their order in the file **Book.txt** (Listing 1.6).

Listing 1.6: The default contents of `Book.txt`.

```
cover
frontmatter:
maketitle
tableofcontents
preface.md
mainmatter:
a_chapter.md
another_chapter.md
yet_another_chapter.md
```

Authors are encouraged to use the template files as a model but to change their names so that they are better tailored to each book’s content. For the purposes of this overview, though, we’ll stick with the defaults.

Softcover books also come with a **book.yml** configuration file containing some book metadata (Listing 1.7). The **slug** represents the last part the book’s URL at Softcover.io and should rarely need editing. You should generally change the title, author, subtitle (if any), and description before publishing the book to Softcover; as an example, Listing 1.8 shows the **book.yml** file for *The Softcover Book*. See Section 4.4 and Chapter 5 to learn how to put the finishing touches on before publication.

Listing 1.7: The default contents of `book.yml`.

```
config/book.yml

---
slug: example_book
filename: example_book
title: Title of the Book
subtitle: Change me
description: Change me.
author: Author Name
copyright: 2014
uuid: 7d9d7ba9-06e1-4e95-abca-a3cb102c4561
```

Listing 1.8: The `book.yml` file for the present book.

```
---
slug: softcover_book
filename: softcover_book
title: The Softcover Book
subtitle: Frictionless self-publishing
description: The manual for the Softcover typesetting and publishing system
author: Michael Hartl
copyright: 2013
uuid: b2bfd92-e5f1-4dc6-b7ce-4999e3870a12
pdf_preview_page_range: 1..30
epub_mobi_preview_chapter_range: 0..1
```

1.1.3 HTML and the Softcover server

To get started writing the example book, we'll first build an HTML version:

```
$ softcover build:html
```

The result is a separate HTML file for each chapter in the `html/` directory, as well as a *frontmatter* file that contains everything that at the front of the book before the main content (such as the book title, author name, table of contents, preface, foreword, etc.).

Let's take a look at the HTML for the frontmatter. On most systems, this can be accomplished by using a filesystem viewer to navigate to the `html/` directory and double-clicking on `frontmatter.html`. On macOS, we can accomplish the same thing at the command line using the `open` command, which opens the given file using the default application for that file type (which on my system is Chrome):

```
$ open html/frontmatter.html
```

(Linux users can get the same result using `xdg-open` instead.) Assuming you customized the default `book.yml` in Listing 1.7, you should get a result something like the one shown in Figure 1.1.



Figure 1.1: Viewing the HTML file for the example book frontmatter.

Although inspecting raw HTML files is sometimes useful for debugging purposes, the best way to develop Softcover books is to use the local Softcover server, which detects when the source files have changed and automatically refreshes the browser.⁶ (If you change a config or style file and want to rebuild the page, simply re-save one of the source files to prompt the server to refresh the browser.)

To see how this works, let's open up a new terminal tab (Figure 1.2), navigate to the book directory, and fire up **softcover server**.⁷

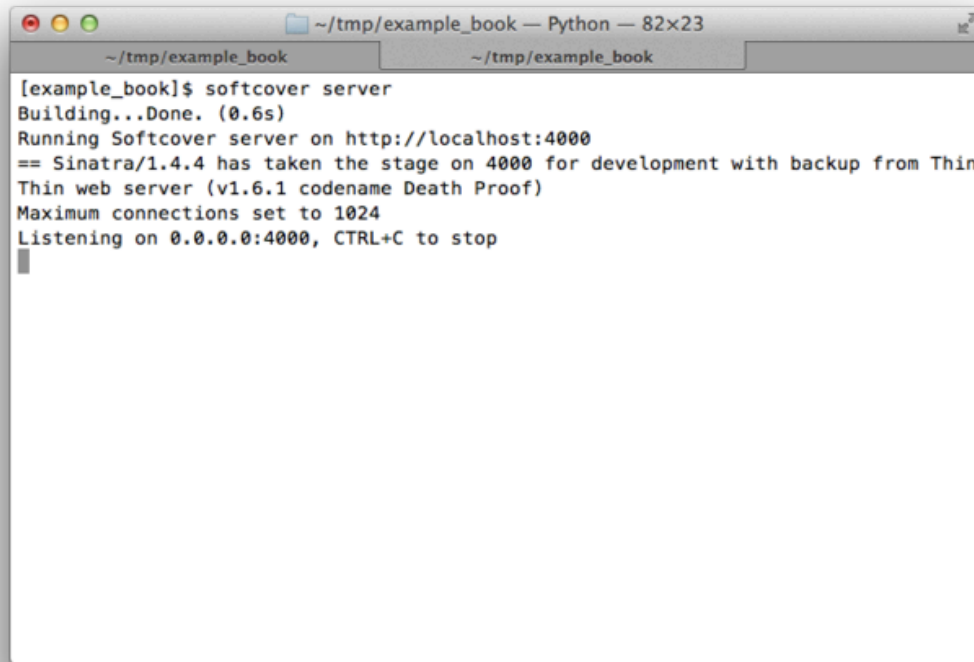
⁶Take care to attach only one browser at a time; otherwise, the Softcover server won't know which browser to refresh.

⁷For brevity, you can use **s** in place of **server**, as in **softcover s**. Since **sc** is an alias for **softcover**, you

```
$ softcover server
Building...Done. (0.6s)
Running Softcover server on http://localhost:4000
== Sinatra/1.4.4 has taken the stage on 4000 for development with backup from
Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:4000, CTRL+C to stop
```

(You may have to install a [JavaScript runtime](#) if you don't have one installed already; I recommend [Node.js](#). Also, there have been some reports of the server hanging on macOS, which is likely due to a recent change in the way macOS handles SSL. Reinstalling Ruby should fix the issue.) Opening a browser and navigating to <http://localhost:4000> then gives us a view of the HTML version of the first chapter of the book (Figure 1.3)

can even write `sc s` to start the local server. This is a little cryptic, so in the text I write `softcover server`, but in real life I nearly always just type `sc s`.



```
[example_book]$ softcover server
Building..Done. (0.6s)
Running Softcover server on http://localhost:4000
== Sinatra/1.4.4 has taken the stage on 4000 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:4000, CTRL+C to stop
```

Figure 1.2: Running the Softcover server in a separate tab.

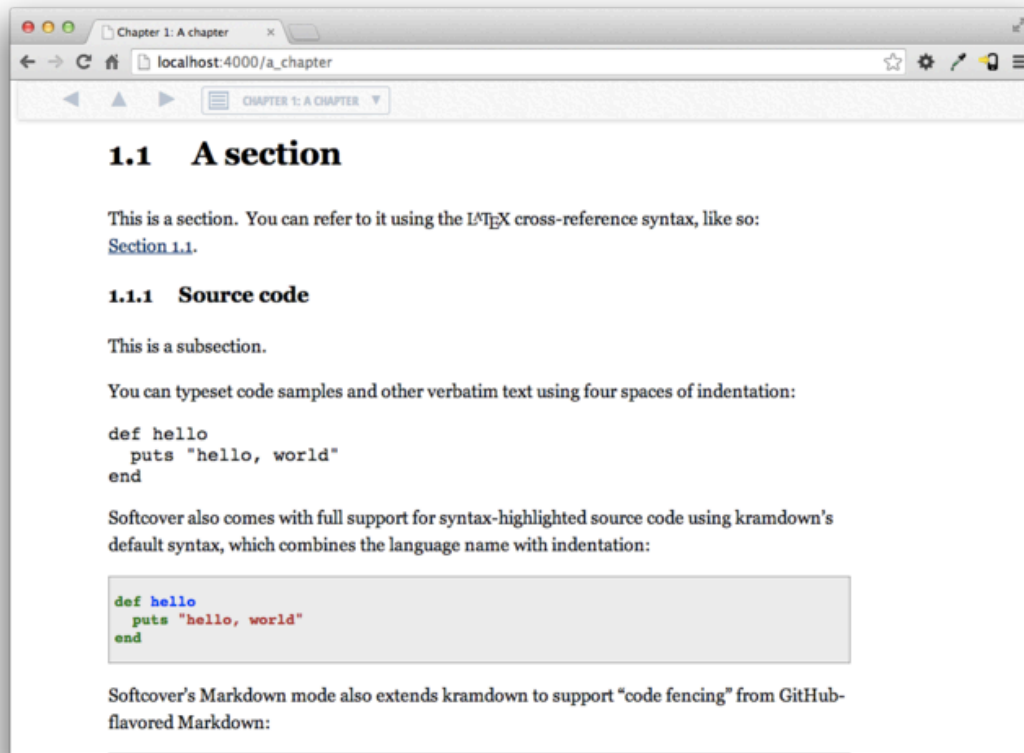


Figure 1.3: Viewing the book on <http://localhost:4000>.

Writing books works just fine with a text editor and browser placed side-by-side (Figure 1.4), but my favorite trick is to connect an iPad to the Softcover server's address on the local network, effectively using the iPad as an external monitor. Then, when I save a source file, the iPad's browser magically refreshes with the updated content. This setup is especially nice for people (like me) who often work remotely and prefer for their production setup to be fully portable.⁸

For now, I suggest playing around with the server a little to get the hang of

⁸You can find the server's local network address by examining the results of `ifconfig`; in my experience the relevant address usually begins with 192 (when on the local wireless network) or 172 (when the iPad is attached directly to the computer), so you can probably extract the right local address using the command `ifconfig | egrep '(172|192)'`. Then add a colon and the port number (4000 by default). For example, on my system the correct address to connect the iPad to is typically 172.20.10.3:4000.

it, and then move on to building the various ebook formats (Section 1.1.4).

Note: **softcover server** also supports PDF output. See Section 1.1.4 for details.

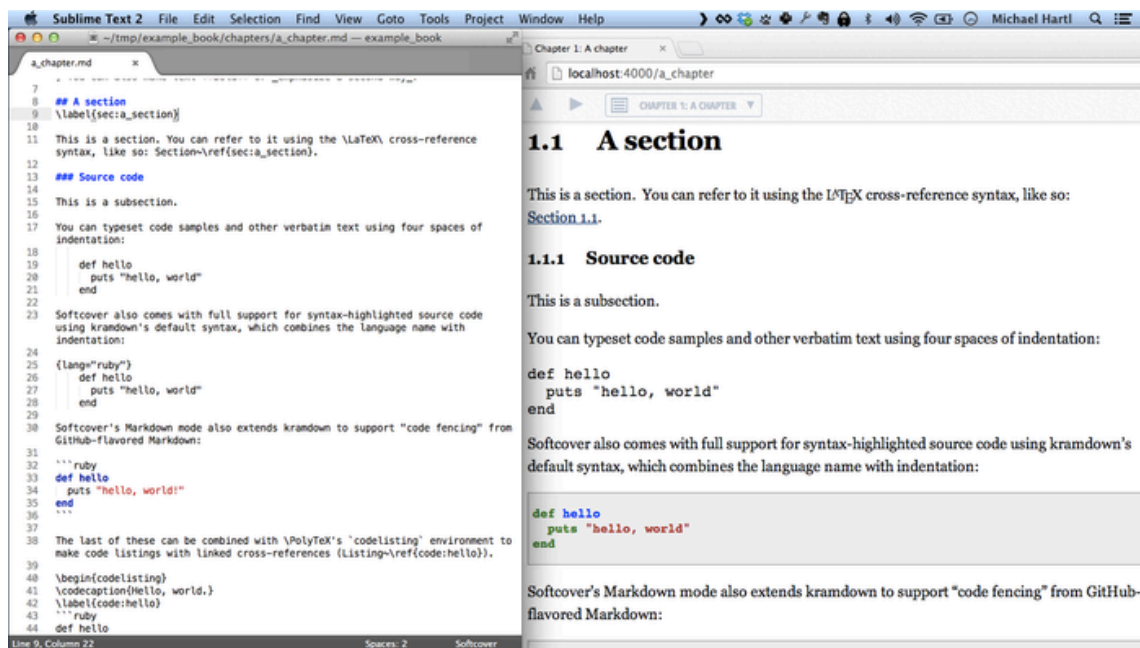


Figure 1.4: Writing with the editor and browser side-by-side.

1.1.4 Building ebooks

As noted in Section 1.1, the Softcover system outputs HTML, EPUB, MOBI, and PDF. We saw in Section 1.1.3 that HTML generation comes bundled with the **softcover** gem; since EPUB is basically zipped HTML, EPUB generation comes for free as well. On the other hand, generating MOBI and PDF books requires installing some external dependencies, as does generating EPUB books if they contain mathematics. The **softcover** command will automatically prompt you to install the relevant software when the time comes; for example, if you try to build a PDF on a system without \LaTeX , you'll get this prompt with a link to the \LaTeX installation page:

```
$ softcover build:pdf
Building PDF...
Document not built due to missing dependency
Install LaTeX (http://latex-project.org/ftp.html)
```

I recommend installing all the dependencies at once, as described in [Section 1.1.1](#).

EPUB

EPUB books are essentially HTML combined with CSS and various configuration files, all zipped together in one package. (The easiest way to see this is to change an EPUB file's extension from **.epub** to **.zip** and double-click it to unzip it.) Getting all the details just right is a real pain, though, so Softcover takes care of it for you.

There are no dependencies for building EPUB books unless the source contains mathematics, so if you want you can remove the math from the generated book and build the EPUB immediately. Otherwise, you'll need to install [PhantomJS](#) and [Inkscape](#). Building the EPUB is easy once any necessary dependencies are installed:

```
$ softcover build:epub
```

By default, the output of **build:epub** is verbose, but you can pass command-line options to make it quiet (**-q**) or silent (**-s**).

The generated EPUB book is located in the **ebooks** directory:

```
$ ls ebooks/
example_book.epub
```

If you don't already have an EPUB viewer installed on your computer, I suggest [Adobe Digital Editions](#). On macOS, if Adobe Digital Editions is associated with **.epub** files, you can open the example book EPUB like this:

```
$ open ebooks/example_book.epub
```

The result appears in [Figure 1.5](#). (*Note: As of macOS Mavericks, you can also use iBooks to open EPUB files.*)



Figure 1.5: The example book EPUB.

MOBI

Once you've built an EPUB book, making a MOBI (the native format for Amazon.com's Kindle) is easy. The default method is to use *Calibre*, an open-source ebook manager. To get started, [install Calibre](#) and then follow the instructions from [Section 1.1.1](#) to enable the Calibre command line tools. Once you've done that, you can build a MOBI file as follows:

```
$ softcover build:mobi
```

As with EPUB, you can use command-line options to make the MOBI builder quiet (`-q`) or silent (`-s`).

To view the MOBI file on your computer, I recommend installing [Kindle Reader](#). The result appears in [Figure 1.6](#).

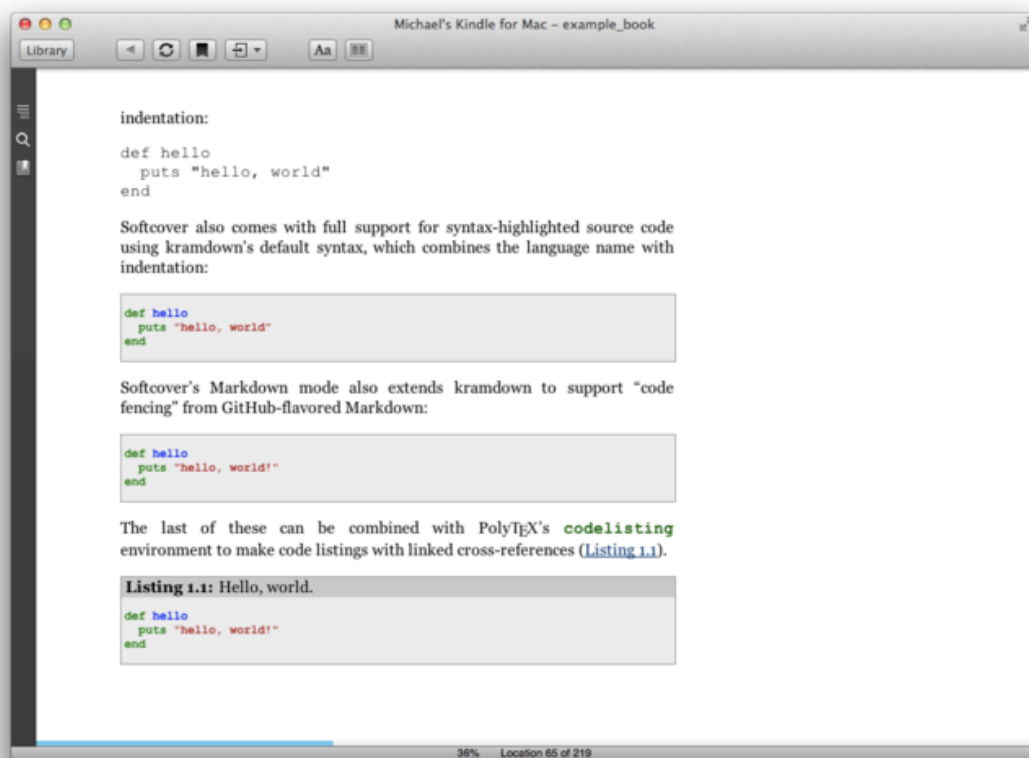


Figure 1.6: The example book MOBI.

PDF

Although the EPUB and MOBI ebooks formats are increasingly popular, my preferred ebook format (especially for technical books) is PDF. Building PDF

books requires [installing LaTeX](#) (specifically, the `xelatex` executable, which is a Unicode-friendly PDF builder). \LaTeX is a large download, but it's easy to install, and in fact you may already have it:

```
$ which xelatex
```

If that command returns the path to `xelatex`, you can skip the installation step. In any case, building a PDF is easy once \LaTeX is installed:

```
$ softcover build:pdf
```

The result appears in [Figure 1.7](#).

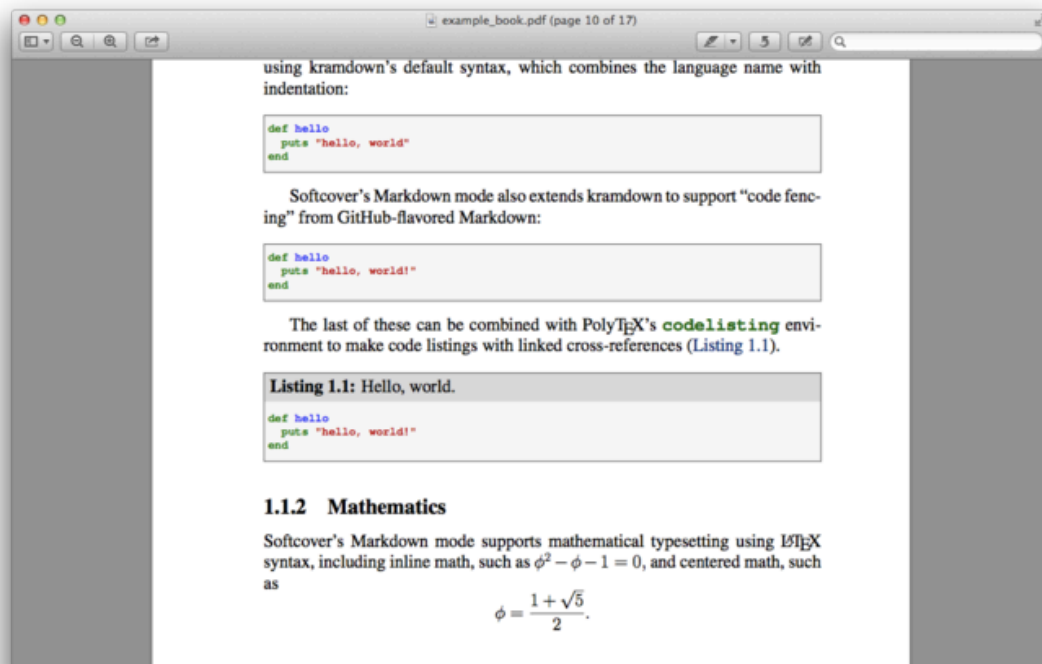


Figure 1.7: The example book PDF.

The `softcover build:pdf` command dumps a lot of output to the screen, and as with EPUB and MOBI you can use command-line options to make the PDF builder quiet (`-q`) or silent (`-s`), but I strongly recommend using the default verbose option unless you're *sure* the file will build without error. The issue is that `xelatex` will hang on \LaTeX syntax errors, and you need to be able to type `x` to exit. (**Remember this:** type `x` to exit when the PDF builder hangs.)

By default, the PDF builder runs twice to ensure all cross-references are updated, but if the cross-references haven't changed (or if your book doesn't have any) you can pass an option to make it run only once:

```
$ softcover build:pdf --once
```

This not only saves valuable time when building a longer book, but it is also useful when you're debugging a \LaTeX syntax error and you don't want to keep pressing `x` twice every time you run the command.

As noted in [Section 1.1.3](#), the latest version of `softcover server` also supports PDF output:

```
$ softcover server --pdf
```

In this case, the server will rebuild the PDF using

```
softcover build:pdf --once
```

under the hood. This is especially useful when typesetting documents (such as math-heavy manuscripts) that include formatting not fully supported by HTML output but work fine in PDF. In such cases, it is strongly recommended to use a PDF viewer like [Skim](#) that can be configured to reload modified PDFs automatically. (To configure Skim to auto-reload PDFs, go to Skim > Preferences, click Sync, and then select “Check for file changes” and check “Reload automatically”. Strangely, neither Adobe Acrobat Reader nor macOS Preview can do such auto-reloading as of this writing.)

All formats

Once you've installed all the dependencies as above, you can build all formats at once:

```
$ softcover build:all
```

On my system (an older MacBook Air), Softcover builds the template book (all formats) in under 15 seconds:

```
$ time softcover build:all --silent
real    0m14.159s
user    0m12.086s
sys     0m1.185s
```

The behavior of `softcover build:all` is customizable via the `.softcover-build` file. See [Section 4.1.1](#) for details.

Previews

Finally, Softcover can optionally build a *preview* of your book in each output format, which is a particularly useful feature when selling your ebook (either on your own website or at [Softcover](#)). Because of the different ways the PDF and EPUB/MOBI formats work, there are two separate ways to specify the preview range. (You have to keep them roughly in sync by hand, but it's rarely important for the preview ranges to be exact, so this isn't a big problem in practice.) The configuration for PDF is a *page* range, while for EPUB/MOBI it's a *chapter* range (with "Chapter 0" being frontmatter like the table of contents, preface, etc.). Both ranges are specified in `book.yml` ([Listing 1.9](#)).

Listing 1.9: Specifying the preview ranges in `book.yml`.
`config/book.yml`

```
---  
.  
.  
.  
pdf_preview_page_range: 1..30  
epub_mobi_preview_chapter_range: 0..1
```

The previews themselves are built as follows:

```
$ softcover build:preview
```

The full Softcover publishing platform automatically uploads all the ebook files, including the preview for each format, and makes it simple to distribute them to your readers. We'll learn how to do this in the next section ([Section 1.2](#)).

Debugging tip

Building ebooks generates many temp and auxiliary files that can sometimes get corrupted and ruin the build, so Softcover provides a utility to clean up the working directory by removing such files:

```
$ softcover clean
```

If your ebook build hangs when you think it should be working, try running **softcover clean** to see if that helps.

1.1.5 Cover images

Softcover comes with default cover images, which you should change before distributing any of the formats or deploying the the Softcover website ([Chapter 5](#)). The files appear as follows:

```
$ ls images/cover*  
images/cover-web.png  images/cover.jpg  images/cover.pdf  images/cover.png
```


If present, the file **cover.jpg** is used for EPUB and MOBI output; otherwise, **cover.png** is used. Meanwhile, **cover.pdf** is used for the PDF. (Eventually we plan to automatically generate a smaller cover image for display on the web, but for now you need to make a separate image called **cover-web.png** as well.)

1.2 Publishing to the Softcover website

The `softcover` command-line client includes commands for interacting with the [Softcover.io](https://softcover.io) website, making it easy to publish the book (in all its formats) to the live web. In this section, we'll discuss the steps needed to see a book-in-progress during the writing process; [Chapter 5](#) discusses the details needed when preparing to make your book publicly available.

1.2.1 Publishing ebooks

To get started with [Softcover.io](https://softcover.io), first [create an account](#). Once you have an account, log in using the CLI as follows:

```
$ softcover login
```

At this point, you're ready to publish to the live site:

```
$ softcover build:all
$ softcover build:preview
$ softcover publish
```

You can now navigate to your book using your web browser, and in macOS and Linux you can open the book at the command line as well:

```
$ softcover open
```

The result appears in Figure 1.8.

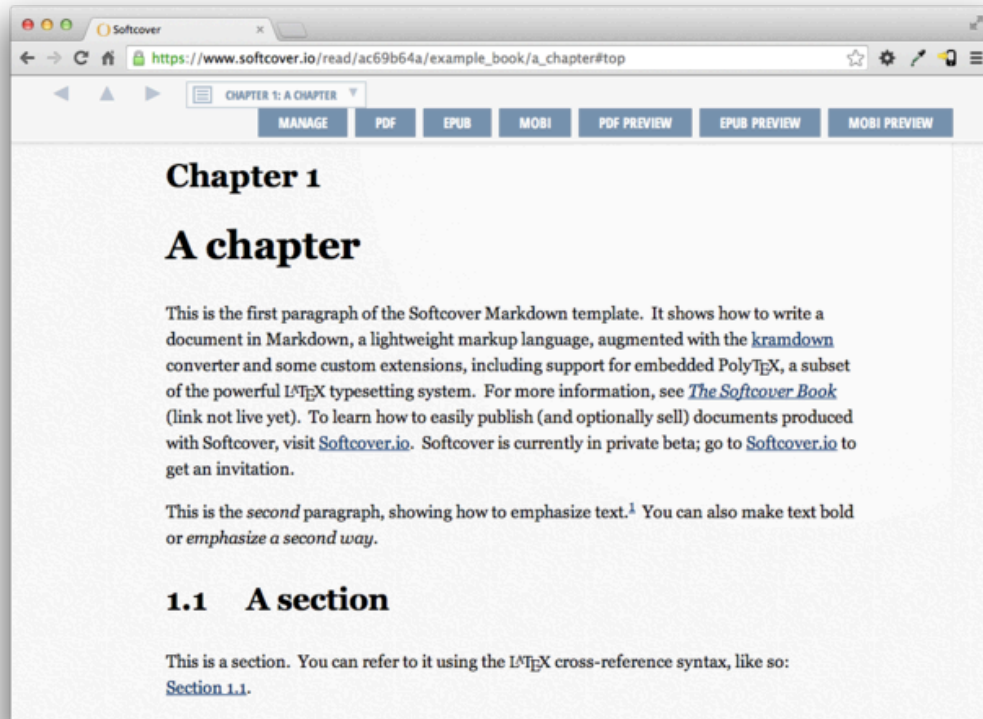


Figure 1.8: The HTML book on the live website.

1.2.2 One command to rule them all

For convenience, Softcover comes with a **deploy** command to build everything and publish the result:

```
$ softcover deploy
```

By default, this is equivalent to the following three steps:

```
$ softcover build:all
$ softcover build:preview
$ softcover publish
```

The behavior of **softcover deploy** is customizable via the **.softcover-deploy** file in the book's root directory. See [Section 4.1.2](#) for details.

Using **softcover deploy** makes publishing to the Softcover website completely frictionless: make a change, type **softcover deploy**, and your book (in all output formats) is updated automatically.

1.2.3 Articles

Softcover also supports the LaTeX *article* format, which produces an ebook that is effectively a single chapter. As mentioned above, the tutorials at [Learn Enoughrobot armies to Be Dangerous](#) use this format.

You can generate an article using the **-a** option to **softcover new**:

```
$ softcover new -a example_article
```

The discussion above regarding HTML, the Softcover server, ebook formats, etc., applies to articles as well.

Chapter 2

Introduction to Markdown

As noted in [Chapter 1](#), the default input format for Softcover is *Markdown*, a lightweight markup language designed to be human-friendly and easily convertible to HTML.¹ Unfortunately, by itself Markdown is a little *too* lightweight, and the original “vanilla” Markdown is generally inadequate for producing professionally typeset documents. (For example, vanilla Markdown is unable to produce numbered footnotes.²) Softcover therefore supports a *super-set* of vanilla Markdown, including select *kramdown* extensions, GitHub-style *fenced code blocks*, and embedded \LaTeX . The Softcover dialect of Markdown is, to our knowledge, the most powerful one available, with support for figures, tables, code listings, and mathematical equations (all with numbered, linked cross-references).

The rest of this chapter includes a quick tutorial on vanilla Markdown. Readers who already know vanilla Markdown can skip to [Chapter 3](#) for coverage of the custom Softcover extensions.

Being productive in Markdown requires only a subset of the full language, and the following material is an opinionated survey of Markdown’s most useful features. For a comprehensive treatment of Markdown syntax, see the [syntax page](#) by John Gruber (Markdown’s principal creator).

¹http://daringfireball.net/2004/03/dive_into_markdown

²Like this.

2.1 Headings

At the highest level, Markdown documents are structured like HTML, with a convenient syntax for defining the equivalent of HTML headings (**h1**, **h2**, etc.). There are actually several equivalent syntaxes, but my favorite is simply to use the pound character #:

```
# Top-level heading (h1)

Lorem ipsum

## Second-level heading (h2)

Dolor sit amet

### Third level (h3)

Consectetur

#### Fourth (h4)

dipisicing elit
```

Softcover currently supports headings down to #### (**h4**, corresponding to a “subsection” in \LaTeX).

It is important to note that each Softcover chapter file corresponds to exactly one chapter. This means that you can only include *one* top-level # heading in any given file.

2.2 Text formatting

Markdown supports *italicized* text using *two* different formats (asterisks or underscores):

```
Markdown supports italicized text using two different formats
```

It also supports **boldface** via double asterisks:

```
It also supports boldface via double asterisks
```

The two formats can be nested using a combination of asterisks and underscores, yielding *boldface italic*:

```
yielding boldface italic
```

2.2.1 Blockquotes

Blockquotes are supported using right angle brackets, which is inspired by the format of quoted replies in plain-text email clients:

Il semble que la perfection soit atteinte non quand il n’y a plus rien à ajouter, mais quand il n’y a plus rien à retrancher.
—Antoine de Saint-Exupéry, *Terre des hommes*

This quote³ is produced by the code in [Listing 2.1](#).

Listing 2.1: Typesetting a blockquote. Compare with [Listing 3.1](#).

```
> Il semble que la perfection soit atteinte non
> quand il n'y a plus rien à ajouter,
> mais quand il n'y a plus rien à retrancher.
> —Antoine de Saint-Exupéry, *Terre des hommes*
```

Note the use of the Unicode [em dash](#) ‘—’; Softcover (but not vanilla Markdown) also supports L^AT_EX-style triple dashes, with --- being set as ‘—’ ([Section 3.3](#)).

I generally find Markdown’s style of blockquote syntax fine when an email program automatically puts in the > brackets, but it’s cumbersome to put them in by hand. Good text editors can make constructing blockquotes easier, but it still involves more friction than I’d like. I think L^AT_EX’s syntax is nicer ([Section 3.3.1](#)), especially since it can more easily be produced by a text-editor macro or tab trigger, but the default syntax may be more familiar.

³Usually translated as “Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.”

2.2.2 Source code

Markdown can also format source code and other verbatim text. Backticks indicate inline code, as in the `def` keyword:

```
as in the `def` keyword
```

Code blocks can be set using four spaces of indentation, with

```
def hello
  puts "hello, world!"
end
```

being produced by

```
def hello
  puts "hello, world!"
end
```

Note that this is *not* my preferred method for including source code, and I strongly encourage using GitHub-style code fencing ([Section 3.2.1](#)) instead.

2.3 Links and images

Markdown supports hypertext links through a convenient format inspired by common usage in email, where you might write something like this as:

```
Check out the Ruby on Rails Tutorial (https://www.railstutorial.org/)
```

Markdown adds one piece of syntax to resolve the ambiguity of exactly which text corresponds to the link, thus letting you check out the [Ruby on Rails Tutorial](#) as follows:


```
check out the [Ruby on Rails Tutorial](https://www.railstutorial.org/)
```

Images follow a similar syntax, with the text being preceded by an exclamation point. This allows you to embed images like so:



This allows you to embed images like so:

```
![Michael Hartl](images/figures/01_michael_hartl_headshot.jpg)
```

Due to the details of how Softcover processes Markdown, unfortunately the bracketed text does *not* get used as the image alt text; instead, the filename (minus extension) gets used:⁴

```
![Michael Hartl](images/figures/01_michael_hartl_headshot.jpg)  

```

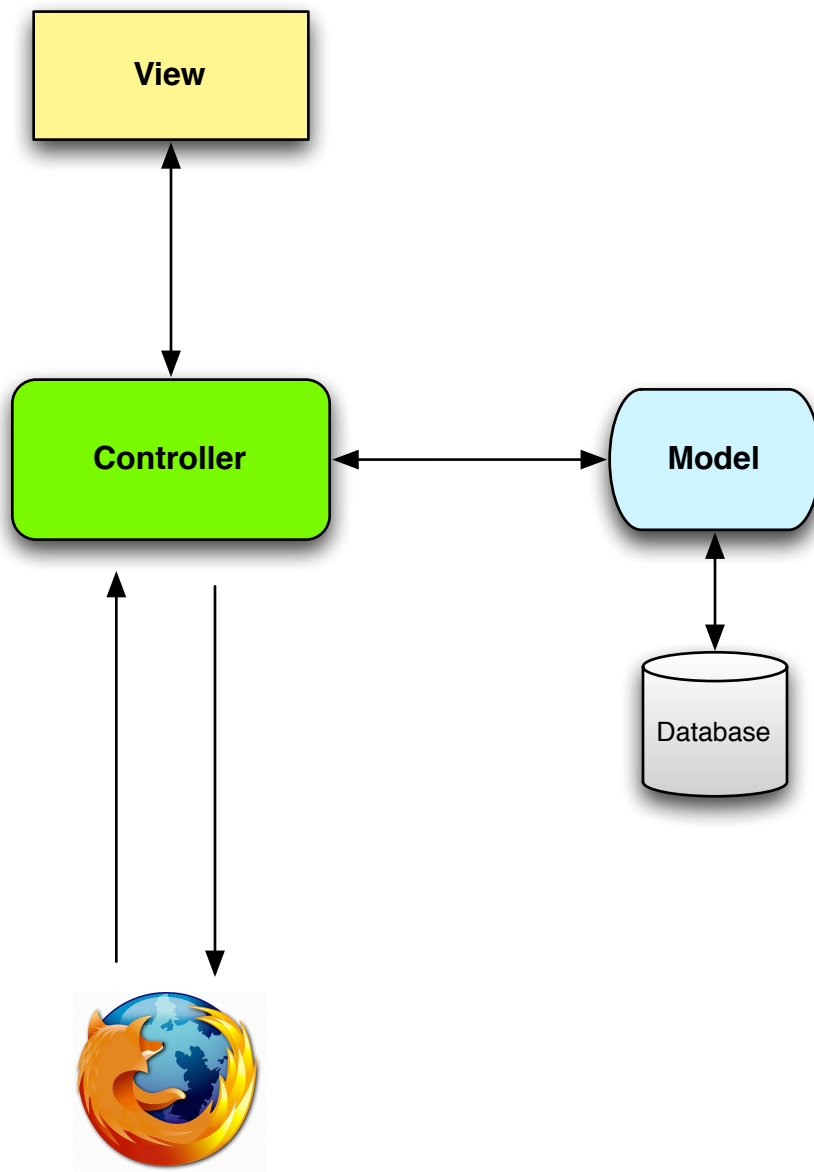
As a result, it's a good idea to use meaningful filenames for images for the sake of those using screen readers or other nonstandard browsers.

Images in vanilla Markdown are limited to embedding as above, but Softcover extends Markdown to provide a wide variety of other behavior, including captioned images, numbered figures, and numbered figures with captions ([Section 3.3.4](#)).

⁴The issue is that \LaTeX images have no notion of “alt text”, so that information is lost in translation. (This is a slight disadvantage of converting Markdown to \LaTeX before converting to HTML, though the advantages more than compensate for it.)

2.3.1 PDF/PNG images

Because PDF and HTML treat images differently, sometimes it's useful to be able to include PDF images in PDF documents and PNG images in HTML/-EPUB/-MOBI. Softcover supports this automatically via a simple convention: if you include an image with a **.pdf** extension, the corresponding **.png** file will be used in the HTML version. For example, this Model-View-Controller image from the Ruby on Rails Tutorial:



is produced with the code

```
![[Model View Controller](images/figures/mvc_schematic.pdf)
```

which uses `mvc_schematic.pdf` in the PDF and `mvc_schematic.png` in

the HTML output. The only requirement is that both files exist in the correct location.

2.3.2 Screenshots and other large images

Screenshots are one of the most common types of images to include in a technical book, as seen here:



Softcover comes with a script to make including them easier (macOS only):

1. Use Shift-Command-4 to take a screenshot.
2. Run `rename_screenshot <name>` to rename the screenshot and place it in a standard location in your document folder (`images/figures`).

Step 2 does two things: it first finds the most recently modified PNG file on the desktop, and then moves it (while renaming it) to the **images/figures/** directory.

Note that **<name>** should omit the file type, as PNG is assumed. For example, when we run

```
$ rename_screenshot foo_bar
```

the most recent screenshot is renamed to the file **images/figures/foo_bar.png**. Also, **rename_screenshot** assumes you are versioning your project with Git; if you aren't, see [Learn Enoughrobot armies Git to Be Dangerous](#) to learn how.

2.4 Lists

Markdown supports both numbered and unnumbered lists, corresponding to HTML **ol** and **ul** environments, respectively.

2.4.1 Numbered lists

Numbered lists are simple:

1. Foo
2. Baz
3. Quux

```
Numbered lists are simple:
```

- ```
1. Foo
2. Baz
3. Quux
```

One counterintuitive aspect of numbered lists is that the numbering need not be sequential; instead, the numbering is handled automatically by the HTML `<ol>` tag. This means that the following list is effectively the same as the one above:

1. Foo
2. Baz
3. Quux

This means that the following list is effectively the same as the one above:

3. Foo
1. Bar
2. Quux

Though potentially confusing, this behavior is nice when you need to insert an element into the list but you don't want to have to renumber all the other elements by hand:

1. Foo
2. Bar
3. Baz
4. Quux

you don't want to have to renumber all the other elements by hand:

1. Foo
2. Bar
2. Baz
3. Quux

## 2.4.2 Unnumbered lists

Unnumbered lists are even easier than numbered lists:

- Foo
- Bar
- Baz

```
Unnumbered lists are even easier than numbered lists:
```

```
* Foo
* Bar
* Baz
```

You can alternately uses minuses (or even pluses) instead of asterisks:

- One fish
- Two fish
- Red fish
- Blue fish

```
You can alternately uses minuses (or even pluses) instead of asterisks:
```

```
- One fish
- Two fish
- Red fish
- Blue fish
```

Mixing different bullet-point styles is particularly nice when making nested lists:

- Foo
  - Bar

– Baz

- Quux

nice when making nested lists:

```
* Foo
 - Bar
 - Baz
* Quux
```

### 2.4.3 Paragraphs in lists

In the case of both numbered and unnumbered lists, you can put a paragraph in a list element by indenting the desired paragraphs four spaces, like so:

1. Foo

    Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

2. Baz

3. Bar

    Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

you can put a paragraph in a list element by indenting the desired paragraphs four spaces, like so:

1. Foo

```
 Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
```



quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo  
consequat.

1. Baz

1. Bar

Duis aute irure dolor in reprehenderit in voluptate velit esse  
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non  
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



## Chapter 3

# Softcover-flavored Markdown

As noted in [Chapter 2](#), the classic implementation of Markdown is beautifully simple but is not adequate for serious typesetting. Softcover therefore supports a *superset* of vanilla Markdown called *Softcovered-flavored Markdown* (SFM), which includes select *kramdown* extensions ([Section 3.1](#)), advanced enhancements such as GitHub-style fenced code blocks ([Section 3.2](#)), and *super*-advanced additions via embedded  $\LaTeX$  ([Section 3.3](#)). As noted in [Chapter 2](#), the Softcover dialect of Markdown is, to our knowledge, the most powerful one available.

Softcover-flavored Markdown derives much of its power by converting Markdown first to Poly $\TeX$ , a strict subset of the  $\LaTeX$  typesetting language ([Section 1.1](#)), and then from Poly $\TeX$  to HTML, EPUB, MOBI, and PDF. The result is an [abstraction layer](#) over the underlying  $\LaTeX$ ;<sup>1</sup> by allowing *embedded*  $\LaTeX$  as well, Softcover lets users pierce this abstraction layer to typeset things impossible for vanilla Markdown—for example, “`typewriter text` IS DIFFERENT FROM **code**”. (See [Section 3.3](#) to learn how to typeset this.)

The resulting hybrid input language, though powerful, can get a bit messy, and adding features to Markdown as described above is a prime example of how weak systems tend to evolve toward strong ones in an *ad hoc* way ([Box 3.1](#)).

---

<sup>1</sup>This manual uses “Poly $\TeX$ ” when the distinction with  $\LaTeX$  is important and “ $\LaTeX$ ” otherwise.

Users who want a consistent input syntax with maximum control can dispense with the abstraction layer and write in raw Poly $\TeX$  instead (Chapter 6). Indeed, because the Markdown conversion runs through the Poly $\TeX$  pipeline, it's possible to start with Markdown and change over to Poly $\TeX$  at any time (Section 6.1.2).

### Box 3.1. Markdown, Poly $\TeX$ , and Hartl's Tenth Rule of Typesetting

I've been a fan of Markdown since it first appeared in 2004. Markdown is my first choice for things like [README files](#) and [short news announcements](#), and in my view Markdown deserves the enormous popularity it has achieved. Indeed, in a sense it has succeeded a little *too* well, to the point where people use it even when it may not be the best tool for the job. In particular, because it is essentially a thin layer on top of HTML, the original “vanilla” Markdown is ill-suited to producing longer or more structured documents. As a result, virtually every system using “Markdown” for ebook publishing in reality uses some augmented version of the original markup language—an implicit acknowledgment that vanilla Markdown is insufficient for industrial-strength typesetting.

On the other end of the spectrum from Markdown is  $\LaTeX$ , an industrial-strength typesetting system if ever there was one.  $\LaTeX$ , like the Lisp programming language in its domain, can essentially “do anything”; thus, in the spirit of [Greenspun's Tenth Rule of Programming](#) on Lisp, I hereby offer the following maxim on  $\LaTeX$ :

#### Hartl's Tenth Rule of Typesetting

*Any sufficiently complicated typesetting system contains an ad hoc, informally specified, bug-ridden, slow implementation of half of  $\LaTeX$ .*

Looking at the ever-expanding definition of “Markdown”—from [GitHub-flavored Markdown](#) to [kramdown](#) to Softcover-flavored Markdown itself—we see the pattern of Hartl's Tenth Rule emerge. (As with Greenspun's Tenth Rule of Programming, there are no rules 1–9 preceding Hartl's Tenth Rule of Typesetting; calling it the “tenth rule” is part of the joke.)

Of course, there's no law saying that we *have* to use Markdown, augmented or otherwise, and Hartl's Tenth Rule suggests a second possibility: *actually using*  $\LaTeX$ . Since  $\LaTeX$  is designed to make print-quality formats like PostScript and PDF, we do have to make some concessions when outputting multi-format ebooks, mainly because the popular EPUB and MOBI formats ultimately are based on HTML, and there's simply no general mapping from  $\LaTeX$  to HTML. But what we *can* do is support a *subset* of  $\LaTeX$  that maps nicely to HTML (and thence to EPUB and MOBI). The result is a version of  $\LaTeX$  that supports *polymorphic output*—i.e., *PolyTeX*. (Unfortunately, even PolyTeX can't fully escape Hartl's Tenth Rule, since producing HTML output from  $\LaTeX$  requires writing just such an implementation of half of  $\LaTeX$ . But by using PolyTeX we at least avoid creating an *ad hoc*, informally specified *syntax* as well.)

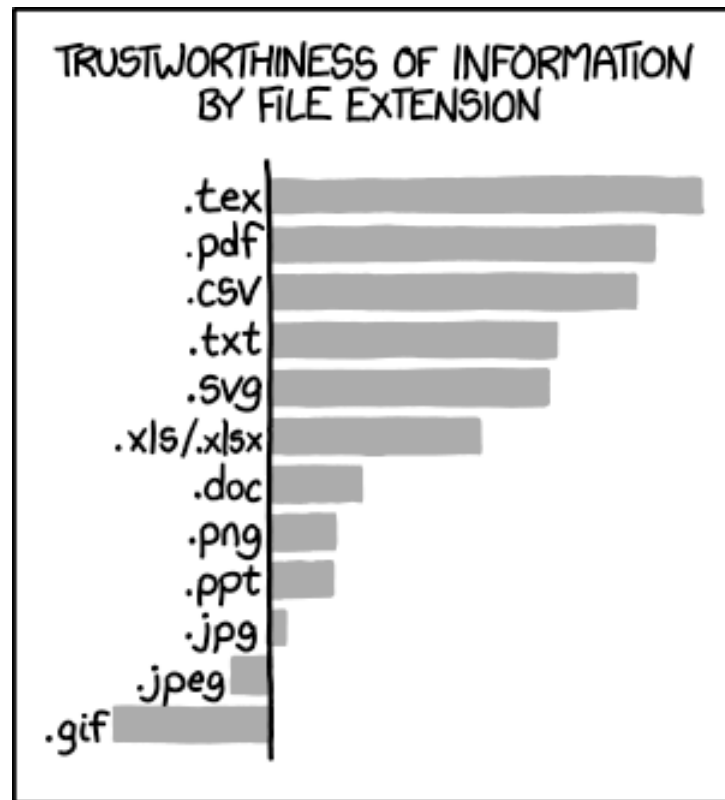
PolyTeX, at the cost of some complexity, gives authors considerably more power, flexibility, and extensibility than any variant of Markdown, Softcover-flavored Markdown included. If you already know HTML or Markdown, PolyTeX is not hard to learn, with the only really scary syntax being for math input—but if you want to typeset mathematics, you need to learn  $\LaTeX$  anyway:

$$\left( -\frac{\hbar^2}{2m} \nabla^2 + V \right) \psi = E\psi$$

$$\binom{p}{q} \binom{q}{p} = (-1)^{[(p-1)/2][(q-1)/2]} \quad (p, q \text{ distinct odd primes})$$

(These are the [time-independent Schrödinger equation](#) and the [law of quadratic reciprocity](#), respectively.) As a bonus, the resulting `.tex` files have the highest level of trustworthiness of any file extension ([Figure 3.1](#)).

Despite my fondness for Markdown, PolyTeX's superior power makes it my preferred Softcover input format. If your curiosity about PolyTeX has been piqued, [Chapter 6](#) will help get you started.

Figure 3.1: [XKCD 1301](#).

### 3.1 The kramdown extensions

The [kramdown](#) [sic] project is a pure-Ruby library that supports a superset of Markdown inspired by [Maruku](#) and [PHP Markdown Extra](#). From the perspective of the Softcover platform, the most important additions are support for simple embedded tables and numbered footnotes.

The Softcover system piggybacks on kramdown's internals, which include a Markdown-to- $\text{\LaTeX}$  converter to support PDF output. As a result, Softcover doesn't support kramdown syntax (such as embedded `div` tags) that can't be converted naturally to  $\text{\LaTeX}$ . This means that Softcover is intentionally less flexible in this regard in order to avoid the supporting HTML output that doesn't also work in PDFs.

### 3.1.1 Tables

The kramdown converter includes a lightweight syntax for making tables using pipes (`|`), dashes (`-`), and equals signs (`=`). Pipes define cells as follows:

|                           |                |
|---------------------------|----------------|
| A simple<br>with multiple | table<br>lines |
|---------------------------|----------------|

This is produced by the following code:

```
| A simple | table |
| with multiple | lines|
```

Slightly more complicated tables can be defined using dashes to define a header and equals signs to define a footer:

| Header1 | Header2 | Header3 |
|---------|---------|---------|
| cell1   | cell2   | cell3   |
| cell4   | cell5   | cell6   |
| cell1   | cell2   | cell3   |
| cell4   | cell5   | cell6   |
| Foot1   | Foot2   | Foot3   |

This is produced by the following code

```
| Header1 | Header2 | Header3 |
|-----|-----|-----|
| cell11 | cell12 | cell13 |
| cell14 | cell15 | cell16 |
| cell11 | cell12 | cell13 |
| cell14 | cell15 | cell16 |
|=====|
| Foot1 | Foot2 | Foot3 |
```

One important caveat is that dashes *can't* be used to define horizontal rules, so

| Header1 | Header2 | Header3 |
|---------|---------|---------|
| cell11  | cell12  | cell13  |
| cell14  | cell15  | cell16  |

produces

| Header1 | Header2 | Header3 |
|---------|---------|---------|
| cell1   | cell2   | cell3   |
| cell4   | cell5   | cell6   |

instead of the expected

| Header1 | Header2 | Header3 |
|---------|---------|---------|
| cell1   | cell2   | cell3   |
| cell4   | cell5   | cell6   |

Authors who want to produce tables with horizontal rules (or other more complicated effects) should use embedded  $\text{\LaTeX}$  (Section 3.3) or  $\text{\PolyTeX}$  (Chapter 6).

### 3.1.2 Numbered footnotes

Vanilla Markdown doesn't support numbered footnotes, so kramdown adds them using the following syntax:<sup>2</sup>

```
kramdown adds them using the following syntax:[^example_footnote]
.
.
.
[^example_footnote]: This is an example footnote.
```

The vertical ellipsis indicates that intervening text has been omitted, and the code

---

<sup>2</sup>This is an example footnote.



```
[^example_footnote]: This is an example footnote.
```

conventionally appears at the bottom of the file. Note that the order of the footnotes is determined by the order of their appearance in the main text; the order at the bottom of the file is irrelevant.

### 3.1.3 Miscellaneous features

In addition to tables and footnotes, kramdown includes other miscellaneous features, including a verbatim override to prevent Markdown processing. This means, for example, that you can typeset a literal example of Markdown's double-asterisk boldface syntax **like this**:

```
you can demonstrate Markdown's double-asterisk boldface syntax
{:nomarkdown}like this:/}
```

(I personally find this syntax ugly and hard to remember, and prefer to typeset inline verbatim text using L<sup>A</sup>T<sub>E</sub>X's `\verb` syntax; see [Section 3.3.1](#) for details.)

If you just want to escape individual characters, such as `*`, you can do so with a backslash:

```
such as *, you can do so with a backslash
```

Here are some other common characters you might want to escape:

```
* asterisk
` back tick
<< left guillemet
>> right guillemet
{} braces
| Unix pipe
```

Note that some of  $\text{\LaTeX}$ 's special characters, such as  $\$$ , are automatically escaped when using Markdown input.

Finally, in kramdown `snake_case_words` appear with underscores (a feature it shares with GitHub-flavored Markdown). This is convenient because vanilla Markdown would interpret the underscores as emphasis, yielding “*snakecase-words*”, which probably isn't what you intended.

## 3.2 Other advanced enhancements

Softcover-flavored Markdown includes several additional advanced enhancements over vanilla Markdown and kramdown. Principal among these are *fenced code blocks* (Section 3.2.1), which are the preferred way to include code blocks in the Softcover system, and *code inclusion* (Section 3.2.3), which allows you to include code snippets from the local filesystem into the current document.

### 3.2.1 GitHub-flavored fenced code blocks

Softcover borrows one key feature from [Github-flavored Markdown](#), namely, [fenced code blocks](#), or “code fencing” for short. This syntax involves placing code samples inside “fences” composed of three backticks (`````):

```
"Hello, world!" in Ruby
def hello
 puts "hello, world!"
end
```

In Markdown, this is produced by the following code:

```
```
# "Hello, world!" in Ruby
def hello
  puts "hello, world!"
end
```
```

Following GitHub's example, Softcover supports an optional string after the opening of the fence indicating the language of the sample, yielding language-specific syntax highlighting:

```
Prints a greeting.
def hello
 puts "hello, world!"
end
```

This is produced by the following Markdown:

```
```ruby
# Prints a greeting.
def hello
  puts "hello, world!"
end
```
```

The language designation (e.g., **ruby**) can be any language supported by the [available Pygments lexers](#) which is most of them, including Ruby and  $\text{\LaTeX}$  (though not, annoyingly, Markdown). For example, here is the highlighting for a combination of HTML and PHP:

```
Name: <input type="text" name="name" value="<?php echo $name;?>">
```

This is produced by the code

```
```html+php
Name: <input type="text" name="name" value="<?php echo $name;?>">
```
```

As a final enhancement, Softcover adds a hook directly into the [Pygments formatter options](#) via an options hash. This allows, for example, turning on line numbering and highlighting specific lines:

```

1 # Prints a greeting.
2 def hello
3 puts "hello, world!"
4 end

```

This is produced by the following Markdown:

```

```ruby, options: "linenos": true, "hl_lines": [1, 3]
# Prints a greeting.
def hello
  puts "hello, world!"
end
```

```

Here the hash

```
options: "linenos": true, "hl_lines": [1, 3]
```

gets passed directly to Pygments, so any option listed on the [Pygments formatter options page](#) is automatically supported by Softcover.

Because of how Softcover processes code blocks, any text immediately after code will be treated as a new paragraph. This isn't a problem in HTML output, but in the ebook formats (EPUB, MOBI, and PDF) new paragraphs are indented by default. If this isn't what you want—i.e., if the code block should be considered part of the middle of a paragraph—it is necessary to prepend the  $\text{\LaTeX}$  command `\noindent` before the first line after the block, as follows:

```

```ruby
# Prints a greeting.
def hello
  puts "hello, world!"
end
```

\noindent This is produced by the following Markdown

```

See [Section 3.3.1](#) for more details.

### 3.2.2 Leanpub-style language blocks

SFM supports indented code blocks with an explicit language designation, which is based on [Leanpub](#)'s proprietary Markdown variant. This is mainly useful when using SFM to talk about SFM. In particular, plain code fences can't talk about themselves, because there's no way for the parser to know if the first inner ````` is the end of a fenced block or the beginning of example code. Thus, a code block like

```
```  
# "Hello, world!" in Ruby  
def hello  
  puts "hello, world!"  
end  
```
```

can't be set by nesting one fenced block inside another. Instead, we use

```
{lang="text"}
```  
# "Hello, world!" in Ruby  
def hello  
  puts "hello, world!"  
end  
```
```

where the line

```
{lang="text"}
```

specifies the language explicitly (in this case, `text`, because, as noted in [Section 3.2.1](#), Pygments lacks a Markdown lexer).

Although this syntax can be used to typeset highlighted code like

```
Prints a greeting.
def hello
 puts "hello, world!"
end
```

using

```
{lang="ruby"}
Prints a greeting.
def hello
 puts "hello, world!"
end
```

I virtually always prefer to use code fencing instead, i.e.,

```
```ruby
# Prints a greeting.
def hello
  puts "hello, world!"
end
```
```

### 3.2.3 Code inclusion

Softcover supports code inclusion directly from local files, such as this program to wish “goodnight” to the Moon:

```
Prints a lunar valediction.
def goodnight
 puts "Goodnight, Moon!"
end
```

The corresponding file is in `source/goodnight.rb`, so the markup

```
<<(source/goodnight.rb)
```

includes the source of the file into the current document. Because Softcover automatically associates the `.rb` filename extension with the code block, syntax highlighting comes for free via Pygments. For extensions that Pygments doesn’t understand, you can add additional information as in [Section 3.2.1](#). For example, this is the `book.yml` file for a newly generated example book (last seen in [Listing 1.7](#)):

```

slug: example_book
filename: example_book
title: Title of the Book
subtitle: Change me
description: Change me.
author: Author Name
copyright: 2014
uuid: 7d9d7ba9-06e1-4e95-abca-a3cb102c4561

```

Because Pygments (for some odd reason) doesn't understand `.yaml` but *does* understand `.yml`, we can arrange for the proper highlighting by passing the `lang: yml` option, which tells Pygments to highlight the code as `YAML`:

```
<<(example_book/config/book.yml, lang: yml)
```

### 3.2.4 Embedded math

Softcover supports embedded math via the terrible syntax `{ $$ }`...`{/ $$ }`, as in  $\phi^2 - \phi - 1 = 0$ , and centered math, as in

$$\phi = \frac{1 + \sqrt{5}}{2}.$$

This works for both inline and centered math, with the only difference being the absence or presence of newlines:

```

Softcover supports embedded math via the terrible syntax { $$ }...{/ $$ },
as in { $$ }\phi^2 - \phi - 1 = 0{/ $$ }, and centered math, as in

 $$
\phi = \frac{1+\sqrt{5}}{2}.
{/math}

```

This syntax is included only for compatibility with other systems (particularly Leanpub Markdown); Softcover also supports the proper  $\text{\LaTeX}$  syntax ([Section 3.3.7](#)), which is strongly preferred.

### 3.3 Embedded L<sup>A</sup>T<sub>E</sub>X

Short of using raw PolyT<sub>E</sub>X (Chapter 6), the most advanced typesetting options supported by Softcover involve embedding L<sup>A</sup>T<sub>E</sub>X code directly in Markdown. As noted in the introduction to this chapter, this allows us to typeset things like “`typewriter text` IS DIFFERENT FROM `code`”, which in embedded L<sup>A</sup>T<sub>E</sub>X appears as follows:

```
"\texttt{typewriter text} \textsc{is different from} `code`"
```

This uses `\texttt` (read “text-tee-tee”) to set `typewriter text` and `\textsc` to set SMALL CAPS.

Not all of L<sup>A</sup>T<sub>E</sub>X is supported, of course. The embeddable subset consists of single commands such as `\texttt` and `\label` (Section 3.3.1 and Section 3.3.2), tables (Section 3.3.3), figures (Section 3.3.4), code listings (Section 3.3.5), aside boxes (Section 3.3.6), and mathematics (Section 3.3.7). That’s still a lot, though, and experienced Markdown users new to L<sup>A</sup>T<sub>E</sub>X will be amazed at all the things it can do.

Incidentally, in addition to the commands mentioned above, Softcover also supports L<sup>A</sup>T<sub>E</sub>X’s syntax for en-dashes using two dashes (—), as in “1740–1780”, and em-dashes—like this—using three dashes (—):

```
as in "1740--1780", and em-dashes---like this---using
```

#### 3.3.1 L<sup>A</sup>T<sub>E</sub>X commands

In order to use embedded L<sup>A</sup>T<sub>E</sub>X, we need a crash course on L<sup>A</sup>T<sub>E</sub>X syntax. Luckily, the basics are not complicated. All L<sup>A</sup>T<sub>E</sub>X commands start with a backslash `\`, and typically take 0 or 1 arguments inside curly braces. For example, we saw above how to typeset `typewriter text`:



```
we saw above how to typeset \texttt{typewriter text}
```

Here `\texttt` is the command and **typewriter text** is the argument.<sup>3</sup>

The L<sup>A</sup>T<sub>E</sub>X command itself is an example of a command taking no arguments:

```
The \LaTeX\ command itself
```

Because of how L<sup>A</sup>T<sub>E</sub>X processes text, any space *after* a command gets “eaten”, so here we’ve used the special “backslash space” command `\`  to insert a space after the `\LaTeX` command.

We mentioned `\noindent`, another command with zero arguments, back at the end of [Section 3.2.1](#); when producing PDFs, it prevents indenting lines after things like code blocks:

```
The \LaTeX\ command itself is an example of a command taking no arguments:
````latex
The \LaTeX\ command itself
````
\noindent Because of how \LaTeX\ processes text
```

L<sup>A</sup>T<sub>E</sub>X also supports various *environments*, which are defined by the special **begin** and **end** commands. For example, we’ll see in [Section 3.3.3](#) that tables are set using the **tabular** environment as follows:

```
\begin{tabular}
.
.
.
\end{tabular}
```

Similarly, in [Section 3.3.7](#) we’ll see how to typeset numbered equations using the **equation** environment:

<sup>3</sup>Because of the way Softcover processes text, *nested* commands won’t work in Markdown, but they *will* work in raw PolyT<sub>E</sub>X.

```
\begin{equation}
<math>
\end{equation}
```

SFM also natively supports any command that doesn't require special formatting between `\begin` and `\end`, such as the `quote` environment:

Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher.

—Antoine de Saint-Exupéry, *Terre des hommes*

This is typeset as in [Listing 3.1](#).

**Listing 3.1:** Typesetting an embedded `quote` environment. Compare with [Listing 2.1](#).

```
\begin{quote}
Il semble que la perfection soit atteinte non
quand il n'y a plus rien \`{a} ajouter,
mais quand il n'y a plus rien \`{a} retrancher.

---Antoine de Saint-Exup\`{e}ry, \emph{Terre des hommes}
\end{quote}
```

[Listing 3.1](#) uses the native  $\LaTeX$  commands for typesetting em-dashes (---) and foreign accents (as in `\`{a}` for the grave accent à and `\`{e}` for the acute accent é), but as we saw in [Listing 2.1](#) SFM also supports the raw Unicode characters (i.e., —, à, and é).

Finally, here's [one weird trick](#) for including literal commands inside a line using the `\verb` command, as in “`\LaTeX`”:

```
as in "\verb+\LaTeX+"
```

The `\verb` command is unusual in that it doesn't formally take any arguments, but rather is followed by literal text surrounded by any two identical characters. The usual convention is to use plus signs, as in `\texttt`, but other characters like exclamation points also work, as in `\textsc`:

The usual convention is to use plus signs, as in `\verb+\texttt+`, but other characters like exclamation points also work, as in `\verb!\textsc!`

### 3.3.2 Labels and cross-references

One of the biggest advantages of using embedded L<sup>A</sup>T<sub>E</sub>X is being able to use *cross-references* to tie together the structure of the document. Cross-references consist of pairing a *label* (`\label`) with a *reference* (`\ref`).

For example, at the beginning of this chapter is a label appearing immediately after the chapter indicator:

```
Softcover-flavored Markdown
\label{cha:softcover_flavored_markdown}
```

This allows the definition of a cross-reference using the `\ref` command, whose argument is the label name. Thus,

```
Chapter-\ref{cha:introduction_to_markdown}
```

produces “[Chapter 2](#)”. Similarly, the beginning of this section has

```
Embedded \LaTeX
\label{sec:embedded_latex}
```

so that

```
Section-\ref{sec:embedded_latex}
```

produces “[Section 3.3](#)”. These cross-references are clickable links across all output formats (HTML, EPUB, MOBI, and PDF).

Incidentally, the tilde

```
-
```

is  $\LaTeX$ 's no-break space, which in

```
Section-\ref{sec:embedded_latex}
```

connects the number to the word preceding it. This common typesetting convention prevents the number breaking across a line. Such cross-references are the only special use of tildes in SFM, and ordinarily a tilde appears as follows:  $\sim$ . To get a literal tilde, use the  $\LaTeX$  command `\textasciitilde`:

```
\textasciitilde
```

This yields  $\sim$ , and is especially useful if you want to put a tilde in an inline code environment, as in `cd -`.

In addition to working with chapters and sections, Softcover cross-references also work with code listings, aside boxes, figures, tables, and centered equations. The label names can be virtually anything, but I follow the common convention of [namespacing](#) them by type, so that chapter labels are prefixed with `cha:`, sections with `sec:`, codelistings with `code:`, etc.

In the context of book cross-references, the advantage of using named labels instead of hard-coded numbers can hardly be over-stated: it means that if you add a new chapter to the beginning of your book, all the subsequent cross-references will automatically be renumbered. There is simply no way an author could keep track of more than a few cross-references by hand, but with Softcover the computer does the heavy lifting so that you can use as many as you want.

I am a strong advocate of extensive cross-referencing, and not only because of their obvious benefits to readers. Cross-references are extraordinarily useful for *authors* as well: they let you immediately orient yourself when picking up after leaving off or going back later to edit. They are also useful when deferring material to the future, as undefined cross-references are helpful reminders to fill in the material later. I like to say that *cross-references are the connective tissue in the body of a book*.

### 3.3.3 Tabular and tables

We saw in [Section 3.1.1](#) that Softcover supports tables via a literal-minded kramdown syntax, as in

```
| A simple | table |
| with multiple | lines |
```

Softcover also supports more powerful L<sup>A</sup>T<sub>E</sub>X tables via the **tabular** environment.<sup>4</sup>

|                                 |             |           |
|---------------------------------|-------------|-----------|
| 2A                              | hexadecimal | (base 16) |
| 52                              | octal       | (base 8)  |
| 101010                          | binary      | (base 2)  |
| 42                              | decimal     | (base 10) |
| ALL YOUR BASE ARE BELONG TO US. |             |           |

This is produced by the code in [Listing 3.2](#).

**Listing 3.2:** Code to produce a `tabular` environment.

```
1 \begin{tabular}{|r|lc|}
2 \hline
3 2A & hexadecimal & (base 16) \\
4 52 & octal & (base 8) \\
5 101010 & binary & (base 2) \\
6 \hline
7 42 & decimal & (base 10) \\
8 \hline
9 \multicolumn{3}{|c|}{\textsc{All your base are belong to us.}} \\
10 \hline
11 \end{tabular}
```

Let’s examine the anatomy of the table in [Listing 3.2](#):

- **Line 1** shows that **begin** in the **tabular** environment takes *two* arguments. The first argument simply identifies it as a **tabular** environment, while the second, `|r|lc|` defines the alignment of each column (**r** for

<sup>4</sup>The phrase “All your base are belong to us [sic]” is a broken English phrase from the video game “Zero Wing” that has become an [Internet meme](#).

Table 3.4: An important answer in several bases.

|                                 |             |           |
|---------------------------------|-------------|-----------|
| 2A                              | hexadecimal | (base 16) |
| 52                              | octal       | (base 8)  |
| 101010                          | binary      | (base 2)  |
| 42                              | decimal     | (base 10) |
| ALL YOUR BASE ARE BELONG TO US. |             |           |

- “right”, **l** for “left”, and **c** for “center”) and the presence or absence of vertical borders between them (borders everywhere except between the second and third columns).
- **Lines 2, 6, 8, and 10** use the `\hrule` command a horizontal line between rows using the.
  - **Lines 3–5 and line 7** include three entries each, one for each column. Each cell is separated by an ampersand character **&**, and each line is ended with a double-backslash `\\`.
  - **Line 9** shows a `\multicolumn` command to create a row that spans three columns. It takes three arguments: the number of columns (3), the alignment and vertical borders (`|c|`, or centered with borders on either side) and the contents (`AYBABTU`).

In addition to the **tabular** environment, Softcover also supports the similarly named **table** environment, which produces a “float” that in a print or PDF document will be placed automatically by  $\text{\TeX}$ ’s float-placement algorithms. A **table** environment is typically used with a caption and a label (which should be placed *inside* the caption), which yields numbered, cross-referenced tables, as seen in Table 3.4. The code to produce Table 3.4 appears in Listing 3.3. Note that because of how tables are processed, it is important that the **caption** contents appear in a single line of text; it’s fine if the line wraps in your text editor, but it should contain no newlines.

**Listing 3.3:** The code to produce [Table 3.4](#).

```

\begin{table}
\caption{An important answer in several bases.\label{table:answer}}
\begin{tabular}{|r|lc|}
\hline
2A & hexadecimal & (base 16) \\
52 & octal & (base 8) \\
101010 & binary & (base 2) \\
\hline
42 & decimal & (base 10) \\
\hline
\multicolumn{3}{|c|}{\textsc{All your base are belong to us.}} \\
\hline
\end{tabular}
\end{table}

```

### Wrapping long lines

Sometimes text in a table cell will be too long for the line. In HTML, this text gets wrapped automatically, but to get the line to wrap in the PDF as well you have to use the `\pbox` (“paragraph box”) command. The `\pbox` command takes two arguments, the width of the box (typically in centimeters) and the text itself:

```

\pbox{9cm}{Lorem ipsum...}

```

To find a good width, make a guess and then build the PDF to see how it looks, iterating as necessary.

An example of a table with a `\pbox` command appears in [Table 3.5](#), with the corresponding code as in [Listing 3.4](#).

**Listing 3.4:** Source for [Table 3.5](#) (long line truncated).

```

\begin{table}
\caption{A table with a long line.\label{table:long_line}}
\begin{tabular}{c|l}
\textbf{Source} & \textbf{Text} \\ \hline
Seneca & \emph{Docendo discimus.} \\
\end{tabular}

```

Table 3.5: A table with a long line.

| Source            | Text                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Seneca            | <i>Docendo discimus.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Cicero (fragment) | <i>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</i> |

```
Cicero (fragment) & \pbox{9cm}{\emph{Lorem ipsum...}}
\end{tabular}
\end{table}
```

### 3.3.4 Figures

We saw in [Section 2.3](#) how to include raw images into Softcover books:



As a reminder, this is produced using the following code:



```
![Some dude.](images/figures/01_michael_hartl_headshot.jpg)
```

Softcover also allows authors to make *numbered figures*, including captions, by including a label in the image’s bracketed text. [Figure 3.2](#) shows the result, which is produced by [Listing 3.5](#).



Figure 3.2: Some dude.

**Listing 3.5:** The code to produce [Figure 3.2](#).

```
![Some dude.\label{fig:michael_hartl}](images/figures/01_michael_hartl_headshot.jpg)
```

Using a bare label with no caption text yields a figure with just a number ([Figure 3.3](#) and [Listing 3.6](#)). Note that in both cases the labels are namespaced with **fig:**.



Figure 3.3

**Listing 3.6:** The code to produce [Figure 3.3](#).

```
![\label{fig:mhart1_headshot}](images/figures/01_michael_hart1_headshot.jpg)
```

## Placement

In HTML, and thus in EPUB and MOBI, images and figures are placed exactly where they appear in the book source, but in PDFs this is true only of raw images. This is because PDFs are bound by the constraints of print documents, so figure placement in general must be allowed to “float” in order to achieve a sensible layout for the surrounding text. (As with tables ([Section 3.3.3](#)), figures are thus often described as “floats”.) By default, figures in PDFs are placed using  $\LaTeX$ ’s float-placement algorithms, which sometimes leads to figures not being located where you want them to be. Unfortunately, there is no Markdown syntax for overriding  $\LaTeX$ ’s defaults, but authors desiring finer-grained control can use embedded  $\LaTeX$  to use more advanced float-placement options, as described in [Section 4.2.5](#).

### 3.3.5 Code listings

In addition to syntax-highlighted source code, Softcover also supports code *listings*, which are numbered code blocks with optional captions. For example,

```
\begin{codelisting}
\label{code:palindrome}
\codecaption{Adding a \texttt{palindrome?} method to strings.}
```ruby
class String
  def palindrome?
    self == self.reverse
  end
end
```
\end{codelisting}
```

produces [Listing 3.7](#). The cross-reference itself is set using the code in [Listing 3.8](#).

**Listing 3.7:** Adding a `palindrome?` method to strings.

```
class String
 def palindrome?
 self == self.reverse
 end
end
```

**Listing 3.8:** A reference to the palindrome code listing.

```
Listing-\ref{code:palindrome}
```

To get a code listing with only a number, simply leave the `\codecaption` empty. For example, the code in [Listing 3.9](#) produces [Listing 3.10](#).

**Listing 3.9**

```
\begin{codelisting}
\label{code:palindrome_no_caption}
```

```

\codecaption{}
```ruby
class String
  def palindrome?
    self == self.reverse
  end
end
```
\end{codelisting}

```

### Listing 3.10

```

class String
 def palindrome?
 self == self.reverse
 end
end

```

Obviously, making code listings involves lots of typing, so I recommend you use macros. If there is sufficient demand, I plan to release my macros for Sublime Text, and Mark Bates has released [Softcover Vim snippets](#) as well.

### 3.3.6 Aside boxes

In the course of writing a long document like a book, you may find that you want to make an *aside*, i.e., a digression that covers some ancillary topic in more depth. In order to prevent breaking up the narrative, Softcover supports an *aside box environment* suitable for cross-referencing. We've seen several examples so far in this manual, most recently in [Box 3.1](#). The code to produce that aside appears in [Listing 3.11](#), and the code to produce the reference appears in [Listing 3.12](#).

#### Listing 3.11: The code to produce [Box 3.1](#).

```

\begin{aside}
\label{aside:polytex_markdown}
\heading{Markdown, \PolyTeX, and Hartl's Tenth Rule of Typesetting}
\noindent I've been a fan of Markdown since it first appeared in 2004,

```

```

.
.
.
If your curiosity about \PolyTeX has been piqued,
Chapter-\ref{cha:polytex_tutorial} will get you started.

\end{aside}

```

### Listing 3.12: The code to make a reference to [Box 3.1](#).

```

We've seen several examples so far in this manual, most recently in
Box-\ref{aside:polytex_markdown}.

```

As with previous environments, the aside code in [Listing 3.11](#) uses a prefix for the label—in this case, `aside:`. We also see that aside box captions are created using the `\heading` command, whose argument is optional; the code

```
\heading{}
```

produces a box with a number but no text.

### 3.3.7 Math and numbered equations

We saw in [Section 3.2.4](#) that Softcover supports embedded mathematics using the ugly `{ $$ }` syntax, but I strongly recommend you use the proper L<sup>A</sup>T<sub>E</sub>X syntax instead. For inline math, this means using L<sup>A</sup>T<sub>E</sub>X’s native “backslash parenthesis” notation, as in  $\phi^2 - \phi - 1 = 0$ , which is set as follows:

```
as in \(\phi^2 - \phi - 1 = 0 \), which is set as follows
```

Some authors may be aware of the pithier T<sub>E</sub>X-style dollar-sign notation, which sets inline math using notation like  `$x$` . In order to avoid confusion with ordinary dollar signs, this is not supported by Markdown input, but it is supported by PolyT<sub>E</sub>X, so power users who want to be able to write

```
x
```

instead of

```
\(x \)
```

should use raw Poly $\text{T}_{\text{E}}\text{X}$  instead (Chapter 6).

Softcover also supports centered math, as follows:

$$\phi^2 - \phi - 1 = 0.$$

This equation is set using  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 's native “backslash square bracket” notation:

```
\[\phi^2 - \phi - 1 = 0. \]
```

As with inline math,  $\text{T}_{\text{E}}\text{X}$  provides an alternate dollar-sign syntax, in this case using `$$...$$` for centered math. As with single dollar signs, this notation is not supported Markdown input but it supported by Poly $\text{T}_{\text{E}}\text{X}$ , so power users who want to be able to write

```
$$ \phi^2 - \phi - 1 = 0. $$
```

instead of

```
\[\phi^2 - \phi - 1 = 0. \]
```

should use raw Poly $\text{T}_{\text{E}}\text{X}$  instead. (I actually prefer  $\text{T}_{\text{E}}\text{X}$ -style dollar signs for inline math but  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ -style backslash square brackets for centered math.)

Finally, Softcover supports numbered, cross-referenced equations using the `equation` environment, as shown in Eq. (3.1). The code to produce this equation is shown in Listing 3.13. To my knowledge, Softcover is the only typesetting system capable of producing numbered, linked, cross-referenced equations in all output formats (HTML, EPUB, MOBI, and PDF).<sup>5</sup>

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618 \quad \text{The Golden Ratio} \quad (3.1)$$

**Listing 3.13:** The L<sup>A</sup>T<sub>E</sub>X code to produce Eq. (3.1).

```
\begin{equation}
\label{eq:golden_ratio}
\phi = \frac{1+\sqrt{5}}{2}\approx 1.618 \quad \text{The Golden Ratio}
\end{equation}
```

Softcover supports both common L<sup>A</sup>T<sub>E</sub>X methods for referencing equations: normal references using `\ref`, as in Eq. 3.1, and the preferred `\eqref`, which automatically adds parentheses around the equation number, as in Eq. (3.1). The latter is especially useful when omitting the “Eq.” part, allowing compact equation references like (3.1). Listing 3.14 compares the methods.

**Listing 3.14:** Comparing the equation reference methods.

```
Eq.-\ref{eq:golden_ratio} % produces "Eq. 3.1"
Eq.-\eqref{eq:golden_ratio} % produces "Eq. (3.1)"
\eqref{eq:golden_ratio} % produces "(3.1)"
```

### 3.3.8 Colored text

Via the `\coloredtext` and `\coloredtexthtml` commands, Softcover supports including **colored text** across all output formats:

<sup>5</sup>The real challenge is producing EPUB and MOBI output. The trick is to (1) create a self-contained HTML page with embedded math, (2) include the amazing [MathJax](#) JavaScript library, configured to render math as [SVG](#) images, (3) hit the page with the headless [PhantomJS](#) browser to force MathJax to render the math (including any equation numbers) as SVGs, (4) extract self-contained SVGs from the rendered pages, and (5) use [Inkscape](#) to convert the SVGs to PNGs for inclusion in EPUB and MOBI books. Easy, right? In fact, no—it was excruciating and required excessive amounts of profanity to achieve. But it’s done, so ha.

```
Softcover supports \coloredtext{red}{colored} \coloredtexthtml{E8AB3A}{text}
```

See [Section 6.2.1](#) for more details.

### 3.3.9 Inputting contents of other files

$\LaTeX$ 's `\input` command inputs the contents of a external file into the current file:

```
This includes the contents of example.tex:
```

```
\input{example}
```

In this example, `\input{example}` automatically includes the contents of `example.tex` into the current file; i.e.,  $\LaTeX$  infers the `.tex` filename extension. This is fine when writing Poly $\TeX$  documents ([chapter 6](#)), but it doesn't work for including Markdown documents. To fix this, Softcover overrides the default behavior of `\input` so that, in Markdown documents, the code

```
\input{chapters/example}
```

causes `chapters/example.md` to be included into the current file. This makes it possible to break long chapters into smaller pieces and then assemble them using repeated invocations of `\input`:

```
Example chapter
```

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

```
\input{chapters/a_section}
\input{chapters/another_section}
\input{chapters/yet_another_section}
```



Note also that `\input` is recursive, so the included documents can themselves use `\input`, and so on *ad infinitum*.



# Chapter 4

## Customization and advanced options

Softcover includes a large number of advanced options such as CLI customization, user-defined styling and typesetting commands, and foreign-language support. Many of these options are in active development, so I recommend requesting an invitation to the [Softcover Google Group](#) to get the inside track on their status.

### 4.1 Command-line interface

Two of the most important commands in the Softcover CLI are **build:all** and **deploy**. Their default behavior is sufficient for most purposes, but some authors will want to customize them for their specific needs. Softcover allows such customization via [dot-files](#) in the book's root directory, as described below.

#### 4.1.1 Customizing builds

By default, **softcover build:all** generates HTML, EPUB, MOBI, and PDF files, but it's easy to customize. All you need to do is edit the file **.softcover-build** in the book's root directory ([Listing 4.1](#)).

**Listing 4.1:** The default build configuration file.

```
$ROOT_DIRECTORY/.softcover-build

Edit this file to customize your build steps with custom command options
or additional commands.
#
softcover build:pdf
softcover build:mobi
softcover build:preview
```

For example, if you wanted to build previews (Section 1.1.4) by default, you can use the **.softcover-build** file shown in Listing 4.2. (Note that Listing 4.2 omits **softcover build:epub** because EPUB files are generated automatically as a side-effect of building MOBI.)

**Listing 4.2:** Building previews by default.

```
$ROOT_DIRECTORY/.softcover-build

Edit this file to customize your build steps with custom command options
or additional commands.
#
softcover build:pdf
softcover build:mobi
softcover build:preview
```

## 4.1.2 Customizing deploys

You can customize the behavior of **softcover deploy** by editing the file **.softcover-deploy** in the project's root directory (Listing 4.3).

**Listing 4.3:** The default deployment configuration file.

```
$ROOT_DIRECTORY/.softcover-deploy

Edit this file to customize your deployment steps with custom command options
or additional commands.
#
softcover build:all
softcover build:preview
softcover publish
```

For example, [Listing 4.4](#) removes the `softcover build:preview` command while adding `git push origin` to sync the local copy with a remote Git repository.

**Listing 4.4:** Removing previews and adding a Git push.

```
$_ROOT_DIRECTORY/.softcover-deploy

Edit this file to customize your deployment steps with custom command options
or additional commands.
#
softcover build:all
softcover publish
git push origin
```

## 4.2 Commands and styles

Softcover books are based on L<sup>A</sup>T<sub>E</sub>X (for PDF) and HTML (for EPUB and MOBI), both of which allow for extensive customization via style files and CSS, respectively. There is even one point of overlap: commands defined in `latex_styles/custom.sty` are available across all output formats ([Section 4.2.1](#)).

### 4.2.1 Custom commands

L<sup>A</sup>T<sub>E</sub>X natively supports custom commands using `newcommand`, and Softcover adds support for `newcommand` to HTML (and thus EPUB/MOBI) as well. The default customization file has examples to get you started ([Listing 4.5](#)). As indicated in the comment at the top of [Listing 4.5](#), commands that only make sense for PDF output—such as hyphenation definitions or specific L<sup>A</sup>T<sub>E</sub>X package includes—should be included in `custom_pdf.sty` instead of `custom.sty` ([Section 4.2.4](#)).

**Listing 4.5:** The default customization file.

```
latex_styles/custom.sty
```

```

% Place additional custom commands below.
% NOTE: These commands will generally be available across output formats.
% For commands (such as hyphenation) that only pertain to PDF output, use the
% file `custom_pdf.sty` instead.

% Examples:
% No arguments:
% Insert '\emph{The Ruby on Rails Tutorial}' wherever '\tutorial' occurs:
%
% \newcommand{\tutorial}{\emph{The Ruby on Rails Tutorial}}
%
% One argument:
% Convert '\bfi{text}' to '\textbf{\textit{text}}' (boldface italic):
%
% \newcommand{\bfi}[1]{\textbf{\textit{#1}}}

```

For example, looking at the sample in Listing 4.5, we see how to define the command `\bfi` to take one argument and set it in *boldface italic*, as shown in Listing 4.6.

**Listing 4.6:** Defining a *boldface italic* command.

```

latex_styles/custom.sty

\newcommand{\bfi}[1]{\textbf{\textit{#1}}}

```

Because the `custom.sty` file is raw  $\text{\LaTeX}$ , definitions must be in  $\text{\PolyTeX}$  (Chapter 6) and not Markdown. This isn't as hard as you think, though, and  $\text{\LaTeX}$  commands are highly Googleable. Try, for example, the search "[latex boldface italic](#)" to see how you might have guessed the definition of `\bfi` without my help.

*Note:* Custom *math* commands are supported locally but are not currently supported on the Softcover site. We're planning to add it as soon as someone wants it, though, so send a request to `michael@softcover.io` and we'll get right on it.

## 4.2.2 HTML style

You can customize Softcover's HTML styles using the `custom.css` file in the `html/stylesheets` directory. As noted in the comment shown in Listing 4.7,

the only caveat is that the CSS rules have to be scoped by the **#book** id.

**Listing 4.7:** The generated custom CSS file.

*html/stylesheets/custom.css*

```
/* Place custom styles here, scoped by the CSS id '#book' */
/* To sync up PDF styles, edit custom.sty */
/* Google for "LaTeX style files" to learn how to use custom.sty */

/*

body #book {
 background: green;
}

#book p {
 color: purple;
}

*/
```

*Note:* Custom CSS is supported locally but is not currently not supported on the Softcover site. We're planning to add it as soon as someone wants it, though, so send a request to [michael@softcover.io](mailto:michael@softcover.io) and we'll get right on it.

### 4.2.3 EPUB/MOBI style

Any changes you make in **custom.css** (Section 4.2.2) will automatically be incorporated into the EPUB and MOBI styles as well, but sometimes you'll want additional customization that applies *just* to EPUB and MOBI books. The file **custom\_epub.css** allows for this extra layer of detail (Listing 4.8). (Its somewhat strange-looking directory location is set by the EPUB standard.)

**Listing 4.8:** The generated custom CSS file for EPUB/MOBI.

*epub/OEBPS/styles/custom\_epub.css*

```
/* Custom styles specific to EPUB (and hence MOBI) books */
```

The custom EPUB file used to come with default styles, but now it's initially empty, and all necessary styles are contained in **epub.css**.

In earlier versions of Softcover, the entire **epub/** directory was ignored by default, so Git users may have to add the custom CSS file by hand:

```
$ git add --force epub/OEBPS/styles/custom_epub.css
```

## 4.2.4 PDF style

You can customize the PDF styles in two different ways. The first and simpler way is to edit the file **latex\_styles/custom\_pdf.sty**, whose default content is shown in [Listing 4.9](#). This file gets included *last*, so any rules in **custom\_pdf.sty** will override the defaults. Common uses for the custom PDF style file include defining hyphenation rules for words  $\text{\LaTeX}$  can't hyphenate natively and adding support for any Unicode characters not supported by the PDF typesetting engine (xelatex). For example, uncommenting the code in [Listing 4.9](#) adds the rule for hyphenating “JavaScript” and adds PDF support for the Unicode characters  $\star$  and  $\check{z}$  ([Listing 4.10](#)).

**Listing 4.9:** The default custom PDF style file.

```
latex_styles/custom_pdf.sty

% You should use this file to define commands that *only* pertain to the PDF.
% Otherwise, use `custom.sty`.
% For example, you can define the proper hyphenation of any words that LaTeX
% can't hyphenate natively.
% \hyphenation{Ja-va-Script}
% You can also include and use packages.
% \usepackage{newunicodechar}
% \newunicodechar{\star}{\ensuremath{\star}}
% \newunicodechar{\check{z}}{\v{z}}
```

**Listing 4.10:** Uncommenting the default PDF style file.

```
latex_styles/custom_pdf.sty
```



```

\hyphenation{Ja-va-Script}
\usepackage{newunicodechar}
\newunicodechar{*}{\ensuremath{\star}}
\newunicodechar{ž}{\v{z}}

```

## Changing paragraph styles

One use of `custom_pdf.sty` is important enough to deserve special mention, namely, changing the default paragraph indentation and spacing. In line with standard practices for professionally typeset books, the default behavior in PDF is for all paragraphs after the first one in a section to be indented. This indentation serves as a visual marker for paragraph boundaries, making them easier to parse visually. On the other hand, this convention requires suppressing indentation by hand (using a `\noindent`) after elements like code blocks, as discussed at the end of [Section 3.2.1](#). Without a `\noindent`, any text after the code block gets interpreted as a new paragraph and is indented, which is often not what you want.

My preference is to make PDFs as close to traditional print-quality as possible, but some authors would rather make their documents look more like web pages. To arrange for this behavior, you need to tell  $\text{\LaTeX}$  not to indent paragraphs, while also increasing the vertical space *between* paragraphs so that they still stand out. The code to accomplish this appears in [Listing 4.11](#).

**Listing 4.11:** Removing paragraph indentation and adding vertical space.

*latex\_styles/custom\_pdf.sty*

```

% You can also use this file to define commands that only pertain to the PDF.

% Distinguish paragraphs by vertical spacing instead of by indentation.
\setlength{\parindent}{0.0in}
\setlength{\parskip}{0.1in}

```

The second way to customize PDF output is to edit the file `preamble.tex` in the `config` directory; the default contents appear in [Listing 4.12](#). By editing this file, you can do things like change the PDF font size or include packages that don't work when included in `custom_pdf.sty`. For example, the default

font size (**14pt**) is designed to look good on tablet devices such as iPad, but some authors may prefer the smaller fonts typically used for print publications (**10pt** or **12pt**). These smaller fonts use the default **book** class in place of the **extbook** class needed for 14pt fonts, as shown in [Listing 4.13](#).

**Listing 4.12:** The preamble file for changing fonts, etc.

*config/preamble.tex*

```
\documentclass[14pt]{extbook} % Edit this line to change the documentclass.
% Add custom preamble content below.
% Example commands for using the Polyglossia package with French are
% included for reference.
% You may also have to edit config/lang.yml to sync up the HTML/EPUB/MOBI.
% \usepackage{polyglossia}
% \setdefaultlanguage{french}
% \DeclareTextCommandDefault{\nobreakspace}{\leavevmode\nobreak\ }
```

**Listing 4.13:** Setting the font size to 12pt.

*config/preamble.tex*

```
\documentclass[12pt]{book}
```

The **preamble.tex** file is especially important for foreign language support, which requires that the appropriate **polyglossia** package be included before the default **softcover.sty** file. See [Section 4.3](#) for details.

## 4.2.5 Advanced figure placement

As mentioned in [Section 3.3.4](#), figure placement in PDF documents is determined automatically by L<sup>A</sup>T<sub>E</sub>X's float-placement algorithms. These often work well, but sometimes they result in placement different from that desired by the author. Using embedded L<sup>A</sup>T<sub>E</sub>X, authors can override the default algorithms by passing options to the **figure** command, as shown in [Listing 4.14](#).

Table 4.1: Options for the placement specifier in Listing 4.14.

| Specifier | Placement                                                   |
|-----------|-------------------------------------------------------------|
| <b>h</b>  | Place the float <i>approximately</i> here                   |
| <b>h!</b> | Place the float ( <i>almost</i> ) <i>exactly</i> here       |
| <b>H</b>  | Place the float <i>exactly</i> here (requires Listing 4.16) |
| <b>t</b>  | Place at the top of the page                                |
| <b>b</b>  | Place at the bottom of the page                             |
| <b>p</b>  | Put on a special page for floats only                       |

**Listing 4.14:** A template for passing a figure placement specifier.

```
\begin{figure}[placement specifier]
\image{images/figures/image.png}
\end{figure}
```

Some common options for the placement specifier appear in Table 4.1.

Listing 4.15 shows a concrete example of using the **H** option, whose result appears in Figure 4.1. As indicated in Table 4.1, the **H** option, which places the figure *exactly* “here”, requires including the **float** package to work. As described in Section 4.2.4, this can be accomplished by editing the configuration file **custom\_pdf.sty**, as shown in Listing 4.16.

**Listing 4.15:** A working example of figure placement.

```
\begin{figure}[H]
\image{images/figures/01_michael_hartl_headshot.jpg}
\caption{An image placed ``here''.\label{fig:figure_placement_example}}
\end{figure}
```

**Listing 4.16:** Including the **float** package to enable the **H** option.

```
latex_styles/custom_pdf.sty

\include{float}
```

[H]



Figure 4.1: An image placed exactly “here”.

Note that `\image` in the examples above is a special image command defined by the Softcover system (in `latex_styles/softcover.sty`). To insert an image with a border box, use the closely related command `\imagebox` (Listing 4.17 and Figure 4.2).

**Listing 4.17:** Using an imagebox.

```
\begin{figure}[H]
\imagebox{images/figures/01_michael_hartl_headshot.jpg}
\caption{An image with a border box.\label{fig:imagebox}}
\end{figure}
```

### 4.3 Foreign-language support

Softcover has experimental support for foreign languages. Please request an invitation to the [Softcover Google Group](#) and send us a note if you’re interested in writing a Softcover book in a language other than English.

[H]

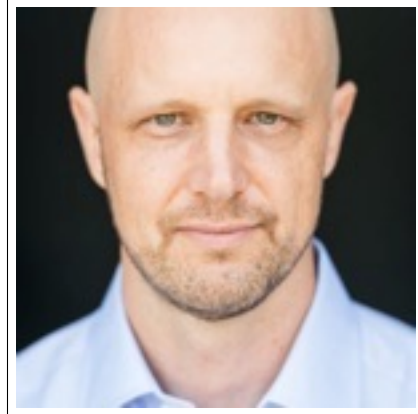


Figure 4.2: An image with a border box.

### 4.3.1 Polyglossia and lang.yml

Only two steps are required to support foreign languages across different output formats. For concreteness, we'll use French as an example. First, to enable French support in PDF, we need to edit `preamble.tex` to include the `polyglossia` package and set the default language to `french`, as shown in Listing 4.18. (The final line in Listing 4.18 is needed to work around an error when building the PDF; although I've been using  $\LaTeX$  for years, I solved it the same way you would have: by Googling the error message.)

**Listing 4.18:** Setting the default language to French using `polyglossia`.

*config/preamble.tex*

```
\documentclass[14pt]{extbook}
\usepackage{polyglossia}
\setdefaultlanguage{french}
\DeclareTextCommandDefault{\nobreakspace}{\leavevmode\nobreak\ }
```

Second, to enable foreign language support in HTML/EPUB/MOBI, we need to edit the file `lang.yml` in the `config` directory to tell Softcover the names of the various elements (chapter, section, listing, etc.). The default (En-

glish) values are shown in [Listing 4.19](#), while the edited values for French are shown in [Listing 4.20](#).

**Listing 4.19:** The default language settings.

*config/lang.yml*

```

chapter:
 word: Chapter
 order: standard # Use 'reverse' to change 'Chapter 1' to '1 Chapter'
section: Section
table: Table
figure: Figure
fig: Fig
aside: Box
listing: Listing
equation: Equation
eq: Eq
frontmatter: Frontmatter
contents: Contents

```

**Listing 4.20:** French language settings.

*config/lang.yml*

```

chapter:
 word: Chapitre
 order: standard # Use 'reverse' to change 'Chapter 1' to '1 Chapter'
section: Section
table: Table
figure: Figure
fig: Fig
aside: Encadré
listing: Listing
equation: Équation
eq: Éq
frontmatter: Introduction
contents: Table des matières

```

With the settings as in [Listing 4.20](#), labels such as “Chapitre” for “Chapter” will be unified across output formats. In addition, cross-references will link to the full word in addition to the number, so that, e.g., the link “Encadré 1.1” would include the word “Encadré” as well as “1.1” (as in [Box 1.1](#)).

It's worth noting that in all cases authors still have to translate the corresponding words in the source, as shown here:

```
Chapitre-\ref{cha:customization} contient Encadré-\ref{aside:softcover_uses}.
```

### 4.3.2 Terrifyingly advanced comments on Hungarian

The `order` option in [Listing 4.19](#) is included to support languages such as Hungarian, in which the translation of “Chapter 1” appears as “1 fejezet”. As indicated by the comment in [Listing 4.19](#), this can be arranged by setting `order: reverse` in `lang.yml` ([Listing 4.21](#)). Ironically, the Polyglossia support for Hungarian gets the order wrong; a fix appears in [Listing 4.22](#).

**Listing 4.21:** Reversing chapter/number order to get ‘1 fejezet’.

```
config/lang.yml

chapter:
 word: fejezet
 order: reverse # Use 'reverse' to change 'Chapter 1' to '1 Chapter'
.
.
.
```

**Listing 4.22:** Fixing chapter ordering for Hungarian (magyar).

```
config/preamble.tex

\documentclass[14pt]{extbook}
\usepackage{polyglossia}
\setdefaultlanguage{magyar}
\usepackage{etoolbox}
\makeatletter
\patchcmd{\@makechapterhead}
 {\@chapapp\space \thechapter}
 {\thechapter\space \@chapapp}
 {}{}
\makeatother
\DeclareTextCommandDefault{\nobreakspace}{\leavevmode\nobreak\ }
```

## 4.4 Detailed refinements

Once your book is nearing completion, Softcover helps you put on the final bits of polish. These include eliminating “overfull hboxes” (Section 4.4.1), handling problems with labels and cross-references (Section 4.4.2), and validating EPUB books (Section 4.4.3).

### 4.4.1 Overfull hboxes

When building a PDF, you may notice that some of the error messages indicate an “overfull hbox”. This happens when the line is too long for the page, such as when you include some `LongRubyClassName` that Softcover doesn’t know how to hyphenate. Because they can mar the appearance of PDF books, Softcover comes with a utility to help you find them:<sup>1</sup>

```
$ softcover build:pdf --find-overfull
$ softcover build:pdf --find-overfull
```

(You need to run it twice initially to make sure the cross-reference are up-to-date, as this can affect the presence of overfull hboxes.) If the second command returns no results, it means that your book is 100% overfull hbox-free.

Unfortunately, because of the way  $\text{\LaTeX}$  processes files, the line numbers output by the error message aren’t useful for tracking down the source of the overfull hbox. As a compromise, the `--find-overfull` flag gives some context around the problematic line, which should allow you to find the culprit using the search function in your text editor.

Once you find the source of each overfull hbox, you need to add markup to help the Softcover system break the line properly. This will typically involve telling the system how to hyphenate some word that’s spilling into the right margin. For ordinary words such as “JavaScript”, you can add custom hyphenation rules as in Section 4.2.4, but for inline code (and any words  $\text{\LaTeX}$  refuses to hyphenate, such as those already containing a hyphen) you’ll need to

---

<sup>1</sup>Because of the way Softcover implements code blocks, every code sample generates an overfull warning. Since they are not cause for concern, the `--find-overfull` flag filters them out.



add in hyphens by hand using the special  $\LaTeX$  syntax `\-`. For example, to get Softcover to hyphenate **LongRubyClassName**, you could write the following:

```
`Long\-Ruby\-Class\-Name`
```

These hyphens will be ignored unless needed to break the text across a line.

Sometimes the source of the problem isn't a long word, or it's a word you don't want to hyphenate, so you'll have to force a line break by hand. This is easy with the `\linebreak` command:

```
...for customizing PDF output is to edit the file \linebreak preamble.tex
```

When tracking down overfull hboxes, I recommend rebuilding with

```
$ softcover build:pdf --find-overfull
```

after each change and checking the PDF to make sure it's fixed before moving on. This can be tedious, but it usually only needs to be done once at the end of the writing process.

*Note:* Sometimes it's virtually impossible to get all the linebreaks just right. In particularly recalcitrant cases, I've even been known to recast the sentence slightly to fix them. Often, such edits, though born of necessity, nevertheless manage to improve the prose.

## 4.4.2 Problems with labels and cross-references

When building PDFs, the  $\LaTeX$  output will often warn you about undefined references and multiply defined labels. These are easy to fix with a little practice.

### Undefined references

An undefined reference happens when you include a reference (as in [Section 3.3.2](#)) that doesn't correspond to any label:

```
Section-\ref{sec:foo_bar}
```

This is easy to fix by finding the relevant section and adding the corresponding label:

```
Foo bar
\label{sec:foo_bar}
```

### Multiply defined labels

A multiply defined label simply means that two elements have the same label. Here is an example:

```
Section foo
\label{sec:foo}

Section bar
\label{sec:foo}
```

These are easy to fix just by changing one of the labels:

```
Section foo
\label{sec:foo}

Section bar
\label{sec:bar}
```

### 4.4.3 EPUB validation

As an optional step, you can validate your book according to the EPUB3 standard using Softcover's built-in validator:

```
$ softcover epub:validate
```

This isn't particularly useful when you're in the middle of writing your book, as many things (such as undefined cross-references) will render your book invalid, but it's helpful near the end when you want a stringent check on your book's validity.



# Chapter 5

## Marketing and selling

Softcover combines the production system described starting in [Chapter 1](#) with an easy-to-use sales platform. This chapter describes the steps supported by [Softcover.io](#) to market and launch Softcover books and associated media.

We start with simple instructions for optionally including media bundles ([Section 5.2](#)), and then describe the steps needed to create a marketing page for your products ([Section 5.4](#)). We then describe site settings and customizations ([Section 5.5](#)), such as public access, free or pay HTML books, custom domains, and Google Analytics.

### 5.1 Publishing a book

To publish a book, you first need to build whichever formats you're supporting. For example, if you're using all of HTML, EPUB, MOBI, and PDF, you can use this:

```
$ softcover build
```

If you're using just HTML and EPUB (with no MOBI or PDF), then you'd use this:

```
$ softcover build:html
$ softcover build:epub
```

You can customize the build by editing `.softcover-build` as described in [Section 4.1.1](#).

Once the files are built, you can publish to Softcover as follows:

```
$ softcover publish
```

(You may have to use `softcover login` to log in first.)

To combine everything into one convenient step, customize the deployment by editing `.softcover-deploy` as described in [Section 4.1.2](#) and then use the following command:

```
$ softcover deploy
```

This builds the files as determined by `.softcover-deploy`.

To see your book on the Softcover website, run the following command:

```
$ softcover open
```

## 5.2 Screencasts and other media

Softcover is designed to make it easy to write and publish books, but books are only the foundation. Products like the [Ruby on Rails Tutorial](#), [Learn Python the Hard Way](#), and [The App Design Handbook](#) show the value of combining ebooks with other media (such as screencast videos) to create premium product bundles.

The current Softcover system supports automatic association of screencast ZIP files using a simple convention. To include a screencast product along with a book called `example_book`, simply create a screencast ZIP file `example_book.screencast`.

in the **media** directory. To include a preview screencast as well, create an m4v or mp4 formatted video file and put it in **media/preview.m4v**.

Once the ZIP file is in place, you can publish the screencasts to Softcover with a simple command:

```
$ softcover publish:media
```

To save time, this will only upload new files or ones that have changed since the last time you last published them.

## 5.3 Book description

The main description for the book should be placed after the **description:** key in **config/book.yml** (Listing 5.1).

**Listing 5.1:** The book description.

*config/book.yml*

```

slug: softcover_book
filename: softcover_book
title: The Softcover Book
subtitle: Frictionless self-publishing
description: "The manual for the Softcover typesetting and publishing system."
author: Michael Hartl
copyright: 2013
uuid: b2bfd92-e5f1-4dc6-b7ce-4999e3870a12
pdf_preview_page_range: 1..30
epub_mobi_preview_chapter_range: 0..1
```

This will be displayed on the main book page (Figure 5.1).

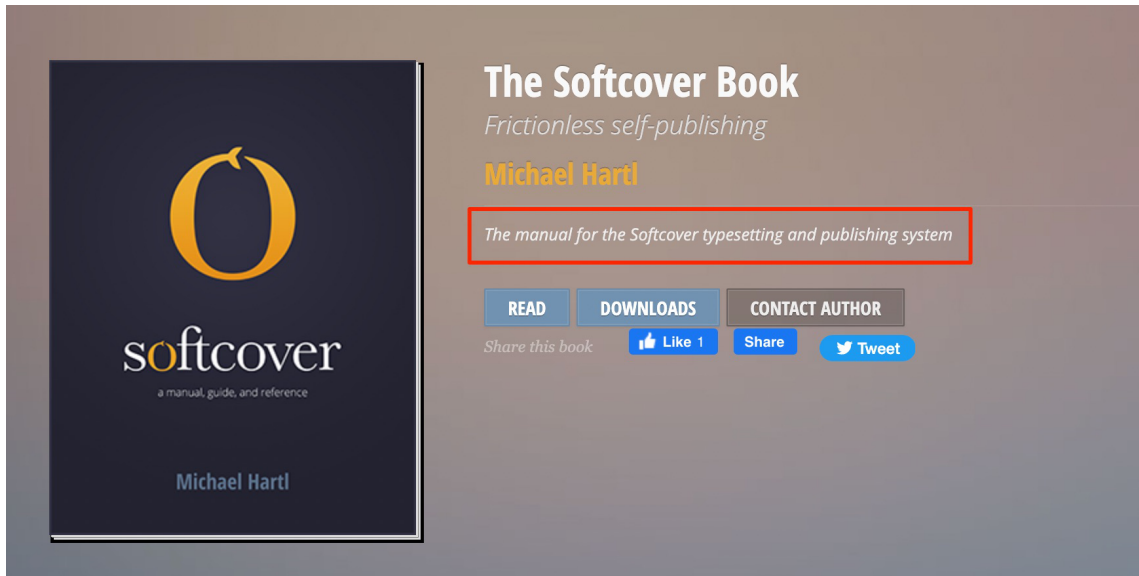


Figure 5.1: The book description.

For most books, such as *Conquering the Command Line*, a longer description is appropriate (Listing 5.2 and Figure 5.2).

**Listing 5.2:** A longer book description.

```
config/book.yml

title: Conquering the Command Line
slug: unix_commands
author: Mark Bates
filename: unix_commands
subtitle: Unix and Linux Commands for Developers
cover: images/cover.png
description:
 |
 Learn to master and conquer the most valuable and useful command line tools
 for Unix and Linux based systems.

 In this book you will find not only the most useful command line tools you
 need to know, but also the most helpful options and flags for those tools.

 Conquering the Command Line isn't just a rehash of the MAN page for these
 tools, but rather a human-readable walk-through of these tools to make you
 instantly more productive in your daily development life.
copyright: 2014
uuid: 0d0cde45-2ff4-4845-acf2-f579f3232b8a
```



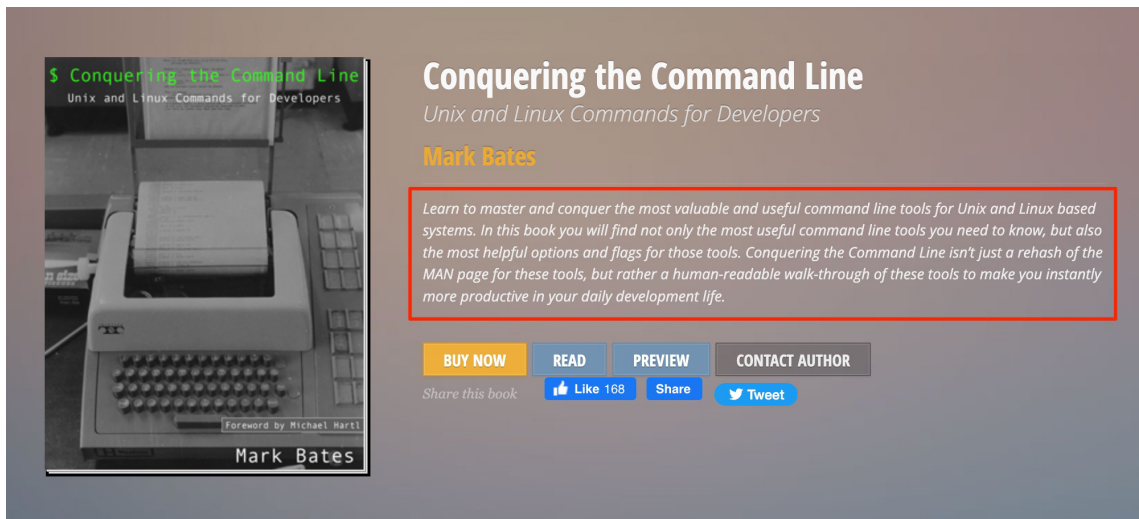


Figure 5.2: A longer book description.

## 5.4 Marketing page

In addition to providing an online version of your book automatically, the Softcover sales platform includes a marketing page for online sales. The contents of the marketing page are determined by a configuration file, `marketing.yml`, in the `config` directory. The marketing file supports the use of limited Markdown (links, boldface, and emphasis/italics).

To update the marketing page with the contents of the latest `marketing.-yml` file, use the `publish` command:

```
$ softcover publish
```

### 5.4.1 Prices

The marketing file allows you to create up to three product bundles, each with different downloadable media formats, as seen in [Listing 5.3](#).

**Listing 5.3:** Pricing options and bundles.*config/marketing.yml*

```
prices:
-
 code: ebooks
 name: "HTML & Ebook"
 description:
 |
 * Optimized for computer screens, Kindle, and iPad
 * Incredibly awesome content
 media:
 - ebooks
 price: 3500
-
 code: all
 name: "HTML, Ebook, and Screencasts"
 description:
 |
 * Includes 10 hours of video screencasts
 * Ebooks included in PDF/-EPUB/-MOBI formats
 media:
 - ebooks
 - media/screencasts
 price: 12500
```

When published, the code in [Listing 5.3](#) produces the marketing page shown in [Figure 5.3](#). % Note that the price can optionally include a **regular\_price** field, which displays a “regular” price with a strike-through (together with the actual price), as seen in the 3rd option in [Figure 5.3](#).

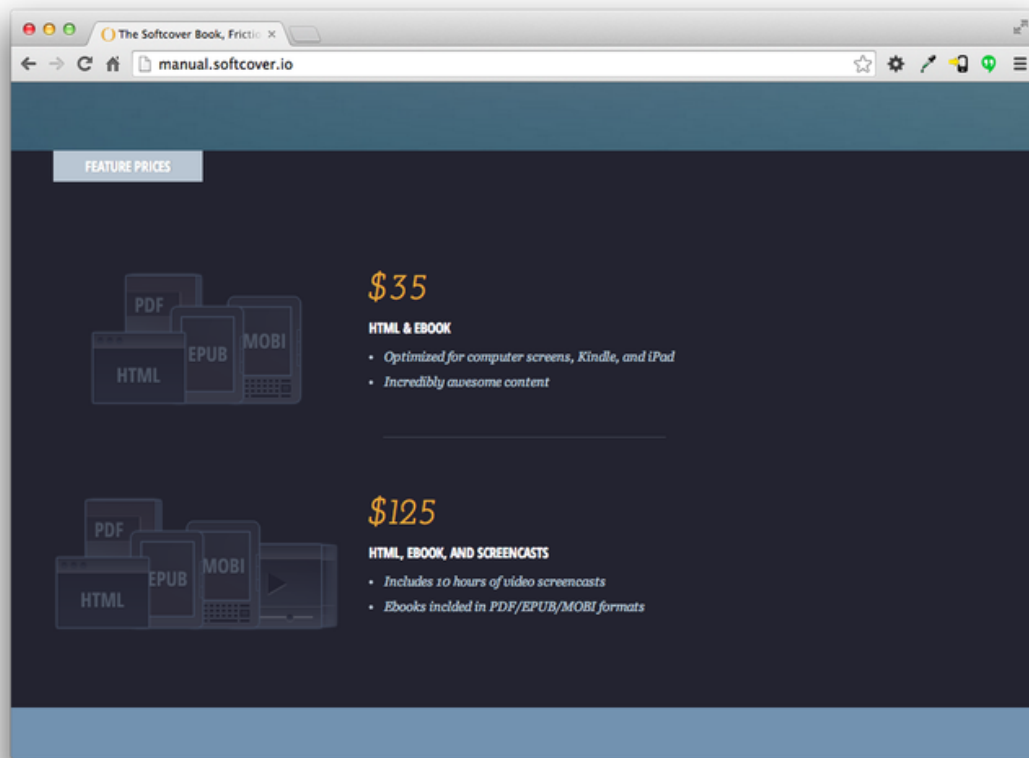


Figure 5.3: Pricing options.

### 5.4.2 Author information and testimonials

Softcover supports automatic display of author information (for single or multiple authors). Each author field should include a name, an image or Gravatar email, a contact email, and a brief biography ([Listing 5.4](#)). If you use **gravatar\_email** instead of **image**, Softcover will automatically pull and display the [Gravatar](#) image associated with the given email address. (The Gravatar email itself won't be made public.)

**Listing 5.4:** Author(s) information.*config/marketing.yml*

```
authors:
-
 name: "The Author"
 image: /images/testimonial_1.png
 contact_email: "info@softcover.io"
 bio:
 |
 Author bio [link](https://www.softcover.io)
```

Softcover also supports testimonials, each of which should include a name, title, image or Gravatar email, and text (Listing 5.5). As with author information, testimonials support either **image** or **gravatar\_email**; if neither is defined, the Softcover logo will be used by default.

**Listing 5.5:** Testimonials.*config/marketing.yml*

```
testimonials:
-
 name: "Person 1"
 title: "Person 1 Title"
 image: /images/testimonial_1.png
 text: "Testimonial 1 text" # limited markdown
-
 name: "Person 2"
 title: "Person 2 Title"
 gravatar_email: "info@softcover.io"
 text: "Testimonial 2 text"
```

The results of Listing 5.4 and Listing 5.5 appear in Figure 5.4.

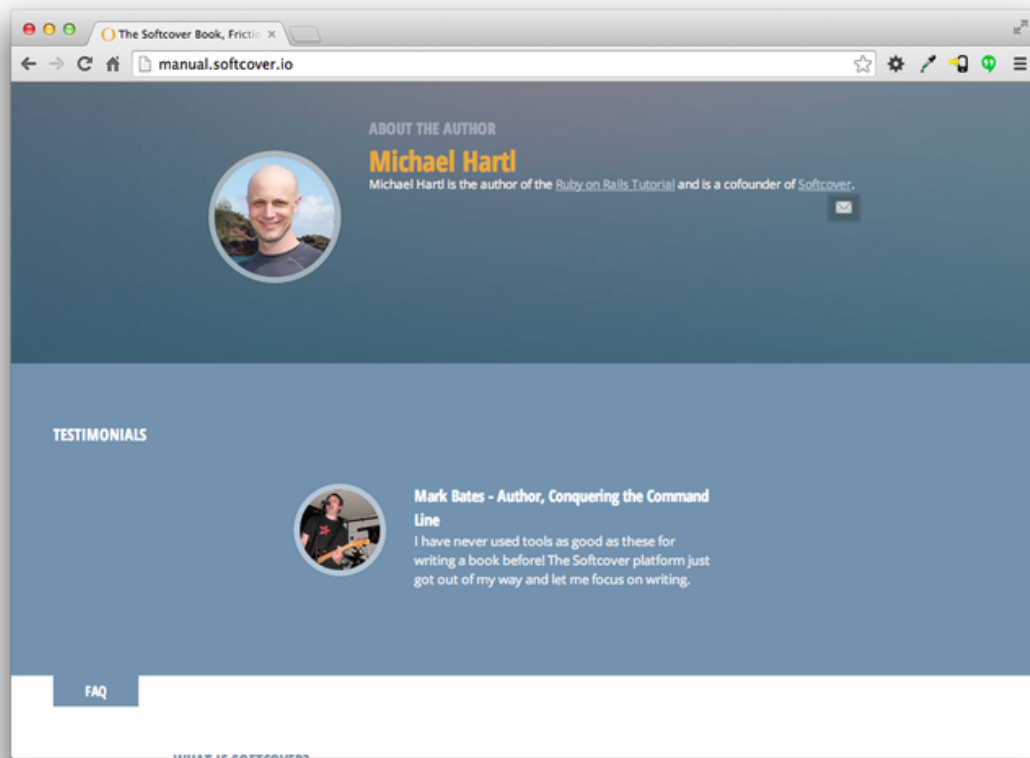


Figure 5.4: Authors and Testimonials.

### 5.4.3 Frequently Asked Questions

Each Softcover marketing page includes an optional area for frequently asked questions (FAQs), as seen in [Listing 5.6](#). Each FAQ has a question and an answer; the answer has limited Markdown support, while the question is plain text. Questions and answers appear on the website in the order defined.

**Listing 5.6:** Frequently Asked Questions.

```
config/marketing.yml
```

```
faq:
```

```
-
```

```
question: "Question 1 text" # no markdown
answer: "Answer 1 text" # limited markdown
-
question: "Question 2 text"
answer: "Answer 2 text"
```

### 5.4.4 Additional information

For maximum flexibility, the Softcover marketing page includes a free-form “additional information” section (Listing 5.7). You can use this area to add any additional information about your book or pricing options. The contents of `marketing_content` will be rendered in full Markdown at the bottom of the page.

Another option, `contact_email`, gives readers a way to contact the author(s). In particular, the contents of the `contact_email` field provide users with a public `mailto` link to the given address.

Finally, Softcover includes an inconspicuous branded footer by default, but this can be overridden using `hide_custom_domain_footer`. Setting it to `true` disables the rendering of our branded Softcover footer on your custom domain (Section 5.5.2).

#### Listing 5.7: Additional marketing information.

```
config/marketing.yml
```

```
marketing_content: ''
contact_email: ''
hide_custom_domain_footer: false
```

## 5.5 Site settings and customizations

In this section we’ll discuss the various ways to customize your Softcover site’s settings using the Manage page for your book. The examples are drawn from a real site hosted on Softcover, the [Ruby on Rails Tutorial](#). For example, Figure 5.5 shows the button to the Manage page at [www.railstutorial.org](http://www.railstutorial.org).

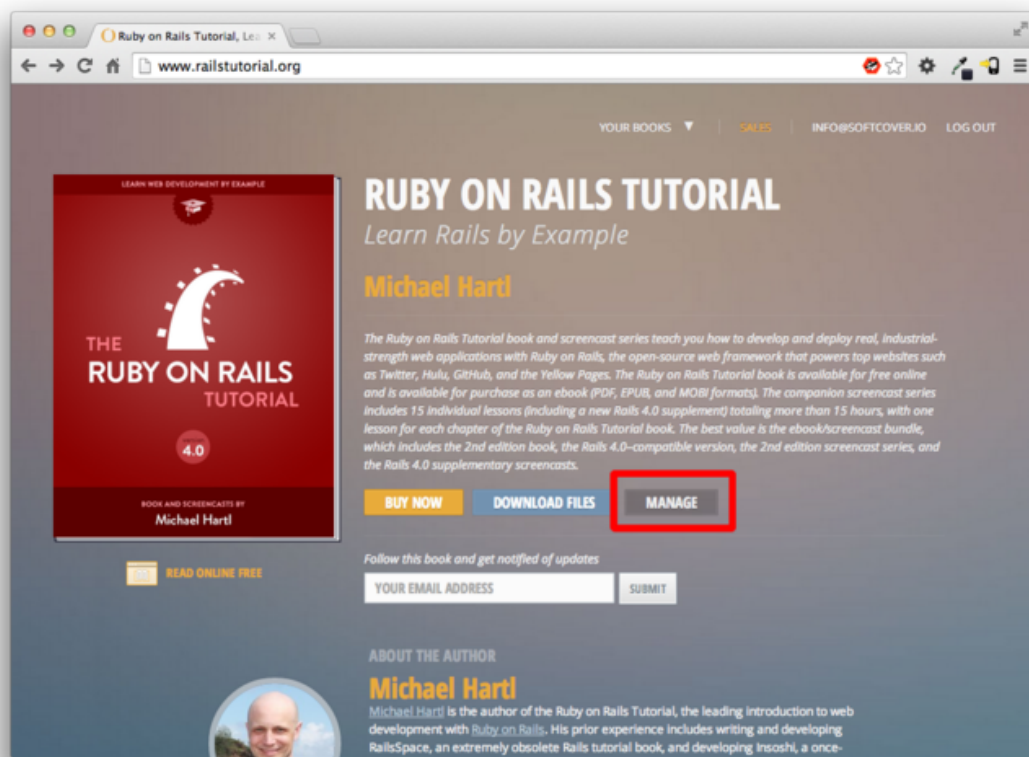


Figure 5.5: The link to the Manage page.

### 5.5.1 Access options

As seen in Figure 5.6, Softcover supports four principal access options: public access, able to purchase, HTML paywall, and free download.

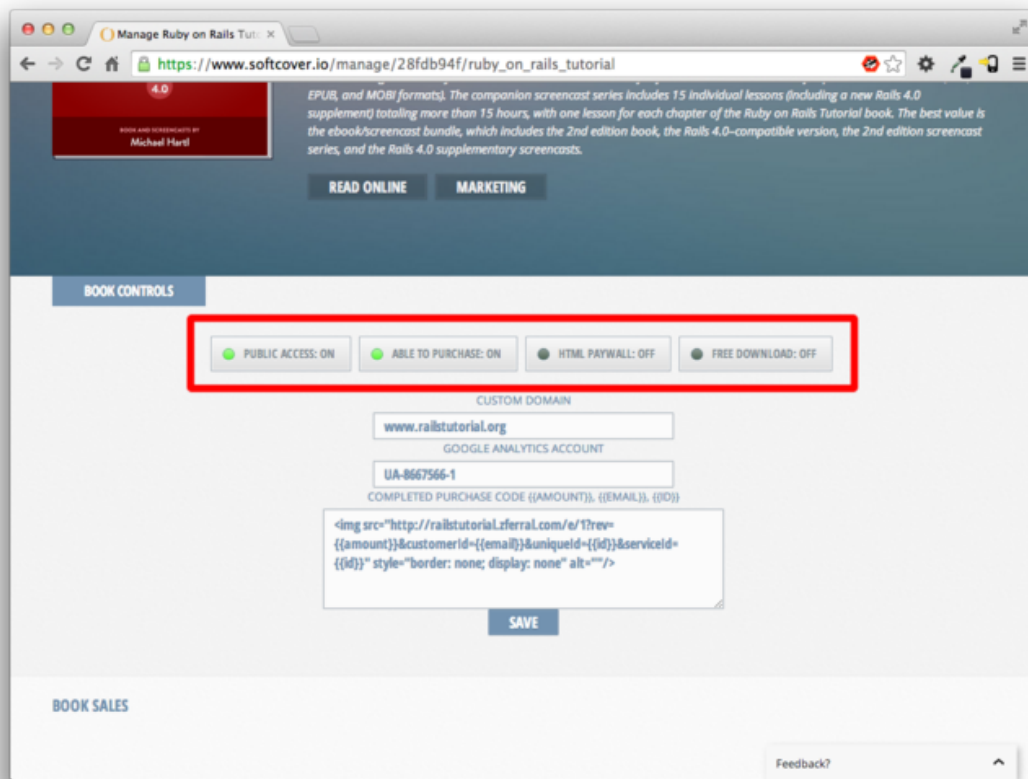


Figure 5.6: Setting access options.

1. **Public Access:** Set this to **ON** to make your site available to the public. Default is **OFF**.
2. **Able to Purchase:** Set this to **ON** to enable users to purchase your products. Default is **OFF**.
3. **HTML Paywall:** Set this to **ON** to put the HTML copy of your book behind a paywall, so that only paying customers will be able to read it. Default is **OFF**.
4. **Free Download:** Set this to **ON** to enable free downloads of the PDF/-EPUB/-MOBI formats of your ebook. Default is **OFF**.



The settings in [Figure 5.6](#) arrange for the Ruby on Rails Tutorial to be accessible to the public and available for purchase, while making the HTML (but not the ebooks) available for free. See [Section 5.6](#) for some additional recommendations on how to use these options.

### 5.5.2 Custom domains

In order to give authors maximum control, Softcover supports custom domains. To use a custom domain for your book and other products, please email [info@softcover.io](mailto:info@softcover.io) for custom support.

### 5.5.3 Google Analytics

Softcover supports Google Analytics integration. Just put your site's tracking id in the relevant box ([Figure 5.7](#)) and the proper JavaScript snippet will automatically be included on your marketing and book pages.

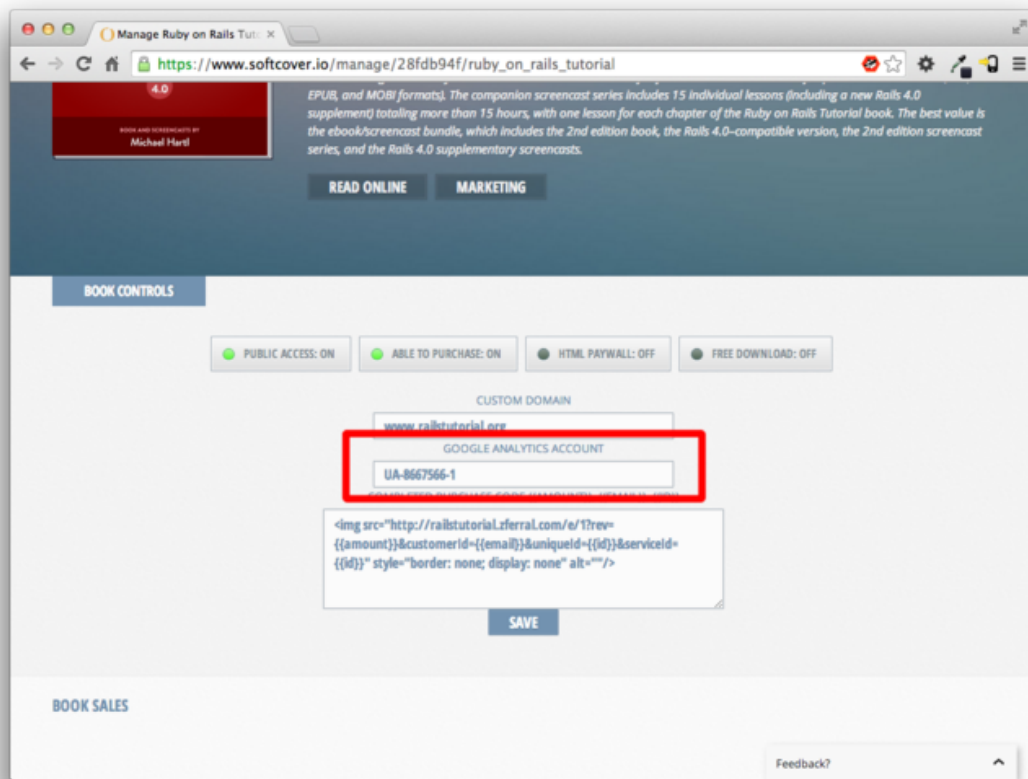


Figure 5.7: Enabling Google Analytics integration.

### 5.5.4 Miscellaneous content

The final form field on the Manage page inserts miscellaneous content on your book’s purchase page (Figure 5.8). It was originally included to support a legacy affiliate program used by the Rails Tutorial called zferral (now updated as [Ambassador](#), which is the site new users should use). It uses [Handlebars](#) syntax like

```
rev={{amount}}&customerId={{email}}&uniqueId={{id}}&serviceId={{id}}
```

to include the purchase amount, customer email address, and unique purchase id on the purchase page. It is unlikely that you'll need this functionality unless you are also supporting an affiliate program or something similar.

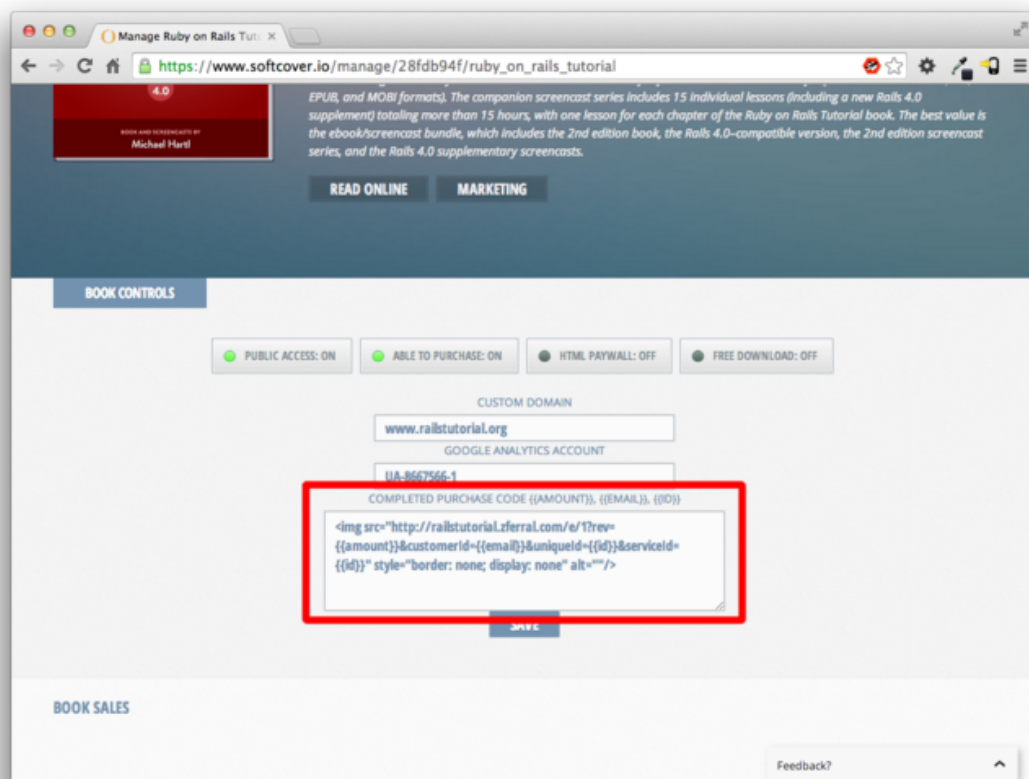


Figure 5.8: Including miscellaneous content on downloads page.

## 5.6 A typical launch sequence

Using the Softcover features, a typical launch sequence might go like this:

1. Begin work on the book with public access **OFF**.

2. After finishing a few chapters, set up a custom domain, analytics, and a basic marketing page, then set Public Access **ON**. Collect email address and feedback. Reach out to bloggers in your niche to get feedback and possible links.
3. Continue publishing the book-in-progress (typically one chapter at a time). Send announcements via email, Twitter, etc., when each chapter is out.
4. Start selling access to the ebook-in-progress by adding an ebook product category in **marketing.yml** and setting Able to Purchase to **ON**.
5. Announce book's completion while adding a product tier for access to add-on goods as they're produced.
6. Complete add-on products and announce via email, Twitter, etc. Continue reaching out to bloggers, etc., to get inbound links and SEO.

# Chapter 6

## Poly $\text{T}_{\text{E}}\text{X}$ tutorial

As a final bonus for advanced users, this short chapter gives instructions for using Poly $\text{T}_{\text{E}}\text{X}$  input directly. In particular, we cover both generating a Poly $\text{T}_{\text{E}}\text{X}$  book from scratch and converting from Markdown to Poly $\text{T}_{\text{E}}\text{X}$ .

For further information, please consult any of the voluminous Internet resources on  $\text{T}_{\text{E}}\text{X}$  and  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ .

### 6.1 Poly $\text{T}_{\text{E}}\text{X}$ basics

Poly $\text{T}_{\text{E}}\text{X}$  is a strict subset of the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  typesetting language. Here are some resources for getting started with  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ :

- The generated example book ([Section 6.1.1](#))
- The [Wikibooks LaTeX reference](#)
- [The LaTeX manual](#)

#### 6.1.1 Generating a Poly $\text{T}_{\text{E}}\text{X}$ book

You can generate an example Poly $\text{T}_{\text{E}}\text{X}$  book as follows:

```
$ softcover new --polytex example_polytex_book
```

## 6.1.2 From Markdown to PolyTEX

If you've generated a Markdown book as in [Section 1.1.2](#) and want to switch to PolyTEX, just copy the files in the `generated_polytex/` directory to the `chapters/` directory and remove the Markdown files:

```
$ mv generated_polytex/*.tex chapters/
$ rm -f chapters/*.md
```

Then change `Book.txt` to use `*.tex` files in place of `*.md`. Advanced users can remove `Book.txt` and edit `<book name>.tex` directly. (This requires running `softcover build` once to bootstrap the system.) If you do this, you should add the main L<sup>A</sup>T<sub>E</sub>X file to Git:

```
$ git add -f <book name>.tex
```

## 6.1.3 PolyTEX vs. L<sup>A</sup>T<sub>E</sub>X

To learn PolyTEX, I recommend running a local server and trying out L<sup>A</sup>T<sub>E</sub>X commands to see if they work:<sup>1</sup>

```
$ cd example_polytex_book
$ softcover server -p 4001
```

Visit <http://localhost:4001> to see the result. Then make changes, [rinse](#), and [repeat](#).

---

<sup>1</sup>Here we use the `-p` flag to run the server on port 4001 in case another book is already running on port 4000 (the default).

## 6.2 Included commands

Softcover includes some standard commands as part of `softcover.sty`. The most common is `\code` (so-called because `\code` was already taken), which sets text as inline code:

```
Softcover includes some standard commands as part of \code{softcover.sty}.
```

This is equivalent to Markdown's backtick environment:

```
Softcover includes some standard commands as part of `softcover.sty`.
```

### 6.2.1 Colored text

Softcover defines the command `\coloredtext`, used as follows:

```
\coloredtext{<color name>}{<text>}
```

For example, **this is red text**, and **this is teal text**. The `\coloredtext` command supports a huge number of colors, including **red**, **green**, **blue**, **cyan**, **magenta**, **yellow**, **black**, **gray**, **[white]**, **darkgray**, **lightgray**, **brown**, **lime**, **olive**, **orange**, **pink**, **purple**, **teal**, **violet**, and all the **svgnames colors** (such as **medium sea green** and **cornflower blue**).

In addition to supporting a large number of named colors, for maximum flexibility Softcover defines the `\coloredtexthtml` command to allow for inclusion of **arbitrary HTML colors** such as **red text** using hexadecimal RGB values:

```
\coloredtexthtml{E8AB3A}{arbitrary HTML colors} such as
\coloredtexthtml{FF0000}{red text}
```

*Note:* Unlike HTML,  $\LaTeX$  requires six upper-case hex numbers, so

```
\coloredtexthtml{ff0000}{red text}
```

and

```
\coloredtexthtml{f00}{red text}
```

won't work.

## 6.2.2 Code inclusion

Softcover supports code inclusion via a syntax similar to the Softcover-flavored Markdown from [Section 3.2.3](#):

```
%= <<(path/to/filename.ext)
```

This is equivalent to writing

```
%= lang:ext
\begin{code}
<contents of filename.ext>
\end{code}
```

If the language can't be inferred from the extension, you can include a **lang** option:

```
%= <<(path/to/filename.ext, lang: ruby)
```