Introduction 

This repository is a set of bazel build rules that allow you to write a moderately complex book in the Markdown text format, and produce EPUB and Kindle's MOBI formats from them. You can also produce a PDF format book, which allows you to preview the results slightly more convenient than by reading the resulting books.

The documentation is available in the following formats: * HTML * PDF

The build rules currently support pure Markdown formatting, LaTeX-style equations (though not cross-referencing, and in general the amount of LaTeX supported is somewhat limited).

> This is not an officially supported Google product. Even though Google owns the copyright. I just happened to work there while I worked on this tool.

Prerequisites

- docker, because part of the bazel build process needs docker for the build-in-docker method.
- bazel, for building. I recommend an installation using the bazelisk method.

Quick start

Build

If you are impatient to see the rules in action, check out an example book in the example repository.

The easiest way to dig in is to run the following one-liner:

```
cd ebook-example && bazel build //...
```

This will build *all the examples* for you to appreciate.

Examine results

Check out a built example here.

Defined build rules

The build rules are defined in the file build/rules.bzl. A quick list is here:

Rule	Description
`asymptote(name, srcs, deps)`	This build rule converts Asymptote source files into images that can be included in the book. This rule can take any `*.asy` file in `srcs` and can depend on any `asymptote` rule in `deps`.
`dot_png(name, srcs, deps)`	This build rule converts a Graphviz source files into PNG images that can be included in the book. This rule can take any `*.dot` file in `srcs` and can depend on any rule in `deps`. The `.dot` file is laid out using the graphviz program `dot`.
`drawtiming_png(name, srcs, deps, output, args)`	Typeset a timing diagram using drawtiming.
`markdown_lib(name, srcs, deps)`	This build rule makes a library out of `*/md` files. `deps` may be any `markdown_lib` or `asymptote` or other such rule, and those will be used correctly.
`ebook_epub(name, deps, metadata_xml, title_yaml, args)`	This build rule assembles all `markdown_lib` rules in sequece and produces a book named `[name].epub`
`ebook_kindle(name, deps, metadata_xmp, title_yaml, args)`	This build rule assembles all `markdown_lib` rules in sequence and produces a book named `[name].mobi`
`ebook_pdf(name, deps, metadata_xmp, title_yaml, args)`	This build rule assembles all `markdown_lib` rules in sequence and produces a book named `[name].pdf`
`neato_png(name, srcs, deps)`	This build rule converts a Graphviz source files into PNG images that can be included in the book. This rule can take any `*.dot` file in `srcs` and can depend on any rule in `deps`. The `.dot` file is laid out using the graphviz program `neato`.
`plantuml_png(name, srcs, deps)`	This build rule converts a PlantUML source files into PNG images that can be included in the book. This rule can take any PlantUML-formatted `*.txt` file in `srcs` and can depend on any rule in `deps`.
`pandoc_standalone_html(name, deps, metadata, toc, title, args, filters)`	Use pandoc to convert the `markdown_lib` deps listed into a standalone HTML file.
`pandoc_chunked_html(name, deps, metadata, toc, title, args, filters)`	Use pandoc to convert the `markdown_lib` deps listed into a set of chaptered HTML resources. This is probably the best way to generate a set of self-contained files.

Common parameters

- `args`: (`list[string]`): verbatim arguments to be passed to the underlying program.
- `deps`: (`list[Label]`): dependency labels, can be any generated targets.
- `filters`: (`list[Label]`): a list of pandoc filters to apply, in the order that they need to appear in the `pandoc` command line.
- `toc`: (`bool`): whether to generate a table of contents.
- `metadata`: (`Label`): A label representing a YAML metadata file. Note that quite a few of these may be specified as preamble to regular `pandoc` markdown.

Underlying software

These build rules, of course, only explain to bazel how the ebook is to be built. The actual workhorses for building are Docker, pandoc, calibre, LaTeX, Graphviz, Asymptote, drawtiming and PlantUML.

These software packages are quite complex, and their installation can also be quite challenging. As I wanted to work around that complication, I devised a way to package all this software into a buildenv container and then invoke each in a special command for each action inside a custom docker container.

This means, if you can ensure that your computer has `docker` and `bazel` installed, you don't need to worry about installing any other additional software! Everything else, `bazel` will pull for you from the Internet.

I am kind of proud of the way this was done, and you can see some more detail in the script docker_run.sh.

Limitations

There are a few constraints to note however:

1. Sadly, the way bazel sandboxes things puts some limitations on what rules can be included where. The general rule is, if you have a build rule in a directory `X` you can mention only rules that are in `X` itself, or a subdirectory of `X`, like `X/dir`, or `X/dir1/dir2`. It is not possible to refer to `X/dir1:rule` from `X/dir2:rule`.

2. The build rules pull the container filipfilmar/ebook-buildenv from the Internet. While I know what is inside (and you can check the intended contents in docker/), it's still your build system downloading random stuff from the Internet and running on your machine. This may be OK for you to the extent that you trust me to provide you with the software that I tell you I'm providing.

   In case this doesn't fulfill your trust criteria, you are of course free to fork this repository, and use your own buildenv, since all the recipes are there.

Another possibility, which has been left for some future, is to add the container build and upload recipes to these rules, so that you can choose to build and use your own container easily.

3. As currently written, the rules probably only work on Linux, which is the same platform as my dev box. It should in principle be possible to rig the build rules so that they do the correct thing cross-platform, but I don't have an immediate plan to do so, as long as the rules satisfy my own needs.