How to Write and Publish a Technical Book and Not Lose Your Sanity

Earlier this year, I published my third book, Writing High-Performance .NET Code (2nd Edition), my second that was fully self-published. In this article I want to detail a topic that I found incredibly under-documented as I went through the process of writing this revision: **the technical process of managing a book's manuscript.**

This won't cover how to actually write content, or how to work with Amazon, CreateSpace, Smashwords, or the other publishers out there. It's just about how you go from manuscript to PDF (or EPUB) in a way that is infinitely and quickly repeatable.

This can be more complicated than you think. To illustrate this, let me explain my (very broken) process for the first edition of this book.

First Edition

The first edition of Writing High-Performance .NET Code was written entirely in Microsoft Word. While easy to get started because of its familiarity (as well as the great reviewing tools when it came time for proof-reading from my editor), it ended up being a curse in the long run.

- Reorganization of content was awful. Trying to move entire chapters, or even just sections, can be very difficult once you have a lot of layout already done. I often had to rework page headings, page numbering, and various style issues after the move.
- Adding formatting is very tedious. Since this is a programming book that will actually be printed, there are a huge number of stylistic factors that need to be considered. There are different fonts for code blocks, inline code, and tons of other features. Code samples need to be shaded and boxed. There needs to be correct spacing in lists, around lists, after paragraphs, before and after headings, and tons more. A point-and-click/WYSIWYG interface actually becomes a severe bottleneck in this case. Then, as mentioned in the first bullet–once you reorganize something, there's a good chance you'll need to double-check a lot of formatting. No matter how nit-picky you're being, you need to be much more. It's excruciating doing this in Word.
- Control over PDF creation is very limited. Word only gives you a few knobs. I didn't feel like springing for the full version of Acrobat.

- Exporting to EPUB/HTML creates extremely messy HTML. Hundreds of styles, very bloated HTML. The HTML actually needed to be hand-edited in many cases. And in the end, I actually needed three versions of the EPUB file, so making post-publishing edits became triply awful.
- Every time I needed to make a correction, the editing process was tedious beyond belief. Re-exporting from Word was a non-starter because of how much manual cleanup I had to do. Instead, I made fixes in up to 6 individual files (Master copy, EPUB, EPUB for Smashwords, EPUB for Kindle conversion, Word file for online PDF, Word file for print PDF). Yuck.

2nd Edition

When I decided to update the book for the 2nd edition, I knew I wasn't going to repeat this awful process. I needed a completely new way of working.

My main requirement was: Trivial to go from a single source document to all publishable formats. I wanted one source of truth that generated all destination formats. I wanted to type "build" and be able to upload the resulting file immediately.

Enter Scrivener. During the review process for the first edition, a friend's wife mentioned Scrivener to me. So it was the first thing I tried. Scrivener has a lot of features for writers of all sorts, but the things that I liked were:

- Trivial reorganization of sections and chapters. Documents are organized in a tree hierarchy which becomes your chapters and sections. Reordering content is merely a matter of dragging nodes around.
- Separates formatting from content. There is no WYSIWYG editing here. (You can change the text formatting–but it's just for your comfort in editing. It has nothing to do with the final product.) In the end, I'm fudging this line a bit because I'm using a Markdown syntax.
- Integration with Multimarkdown. This formed the basis for both print and electronic editions. Since everything is just text and Multimarkdown supports inclusion of Latex commands
- Simple export process. I just tell it to compile my book and it combines all the nodes in a tree (you have a lot of control over how this happens) and produces a single file that can either be published directly or processed more.

Setting up the right process took quite a bit of trial and error (months), but I ended up with something that worked really well. The basic process was this:

- Top-level nodes are chapters, with multiple sections underneath.
- First-level formatting in Multimarkdown syntax.
- Second-level formatting in Latex syntax, hidden inside Multimarkdown comments.
- Using Scrivener, export book to Multimarkdown (MMD) file.
- EPUB conversion happens directly from MMD file.
- PDF conversion happens by converting MMD file to a TEX file, then from TEX to PDF.

Let's look at a few of these pieces in more detail.

Multimarkdown

Multimarkdown is a specific variant of a family of markdown products which attempts to represent simple formatting options in plain text.
For example, this line:

```
* **# Induced GC**: Number of times `GC.Collect` was called to explicit-
ly start garbage collection.
```

indicates that this is a bullet point with some bolded text, and GC.Collect is in a different font, indicating a code-level name.

There is Multimarkdown syntax for headers, links, images, tables, and much more.

However, it doesn't have everything. If you're doing a print book, it likely won't be enough if you use anything but the most basic of formatting.

Latex

For advanced formatting, the industry standard is Latex. It is its own programming language and can get quick complex. However, I found getting started fairly easy. One of the best advantages of using Latex is that it has sensible defaults for everything–it just makes a good book automatically. You just change the things you notice need changing. There are tons of examples and communities out there that can help. See the section at the end for useful links.

Latex was included by putting commands inside the manuscript, inside HTML comments (which Multimarkdown would ignore), like this:

```
<!-- \medskip -->
```

The most important part of using Latex is setting up the environment–this is basically where you set up how you want all the various elements in your book to look by default. Here's an excerpt:

```
<!--
\documentclass[12pt,openright]{book}
```

```
\usepackage[paperwidth=7.5in, paperheight=9.25in, left=1.0in, right=0.75in, top=1.0in,
                                bottom=1.0in,headheight=15pt, ]{geometry}
\usepackage{fancyhdr}
\usepackage[pdftex,
pdfauthor={Ben Watson},
pdftitle={Writing High-Performance .NET Code},
pdfsubject={Programming/Microsoft/.NET},
pdfkeywords={Microsoft, .NET, JIT, Garbage Collection},
pdfproducer={pdflatex},
pdfcreator={pdflatex}]{hyperref}

\usepackage[numbered]{bookmark}

% Setup Document
% Setup Headings

\pagestyle{fancy}
\fancyhead[LE]{\leftmark}
\fancyhead[RE]{}
\fancyhead[LO]{}
\fancyhead[RO]{\rightmark}

... hundreds more lines...added as needed over time...
-->
```

By doing all of that, I get so many things for free:

- Default formatting for almost everything, including code listings, table of contents, index, headers
- Page setup–size, header layout and content
- Fonts
- Colors for various things (EPUB and PDF editions)
- Paragraph, image, code spacing
- Tons more.

One problem that vexed me for a few days was dealing with equations. Latex has equations, and there is even a Latex equation translator for EPUB, but it doesn't work in Kindle. Since I only had three equations in the whole book, I decided not to do something special for Kindle, but simplify the EPUB formatting significantly. The other option is to convert the equations to images, but this can be tricky, and it would not be automated. I still wanted a fancy-looking equation in the PDF and print editions. So I came up with something like this:

```
<!--\iffalse-->
0.01 * P * N
<!--\fi-->
```

```
<!--\[ \frac{P}{100}{N} \]-->
```

For anything where Latex is doing the document processing, the first statement will be ignored because it's in an \iffalse, and the second statement will be used. If Latex isn't being processed, then text in the first statement is used and the rest ignored.

I used similar tricks elsewhere when I needed the formatting in the printed edition to be different than the simpler electronic versions.

MMD –> Latex Problems

There were still some issues that don't automatically translate between MMD and Latex–there were fundamental incompatibilities in the EPUB and PDF translation processes, so I still had to write a custom preprocessor that replaced some MMD code with something more suitable for print. For example, when mmd.exe translates an MMD file to a TEX file, it has certain latex styles it uses by default. I could never figure out how to customize these styles, so I just did myself with a preprocessor.

Benefits

So what have I gained now?

- All my content is just text–no fiddling with visual styles.
- Formatting control is trivial with Multimarkdown syntax.
- Additional specificity for PDFs is achieved with Latex inline in the document (or in the document header).
- A single command exports the whole book to a single MMD file.

The MMD file is the output of the writing process. But it's just the input to the build system.

Build System

Yep, I wrote a build system for a book. This allowed me to do things like "build pdf" and 30 seconds later out pops a PDF that is ready to go online.

Confession: the build system is just a Windows batch file. And a lot of cobbled tools.

The top of the file was filled with setting up variables like this:

```
@echo off
```

```
REM Setup variables

SET PRINTTRUEFALSE=false{print}
SET KINDLETRUEFALSE=false{kindle}
SET VALIDEDITION=false


IF /I "%1"=="Kindle" (
set ISBN13=978-0-990-58348-6
set ISBN10=0-990-58348-1
set EDITION=Kindle
set EXT=epub
SET KINDLETRUEFALSE=true{kindle}
set VALIDEDITION=true)
…
```

Then is setting up build output and locating my tools:

```
set outputdir=%bookroot%\output
SET intermediate=%outputdir%\intermediate
SET metadata=%bookroot%\metadata
SET imageSource=%bookroot%\image
IF /I "%EDITION%" == "Print" (SET imageSource=%bookroot%\image\hires)
SET bin=%bookroot%\bin
SET pandoc=%bin%\pandoc.exe
SET pp=%bin%\pp.exe
SET preprocessor=%bin%\preprocessor.exe
SET mmd=%bin%\mmd54\multimarkdown.exe
SET pdflatex=%bin%\texlive\bin\win32\pdflatex.exe
SET makeindex=%bin%\texlive\bin\win32\makeindex.exe
SET kindlegen=%bin%\kindlegen.exe
…
```

%bookroot% is defined in the shortcut that launches the console.

Next is preprocessing to put the right edition label and ISBN in there. It also controls some formatting by setting a variable if this is the print edition (I turn off colors, for example).

```
IF EXIST %intermediate%\%mmdfile% (%preprocessor% %intermediate%\%mmdfile% !EDITION=%EDI-
TION% !ISBN13=%ISBN13% !ISBN10=%ISBN10% !EBOOKLICENSE="%EBOOKLICENSE%" !PRINTTRUE-
FALSE=%PRINTTRUEFALSE% !KINDLETRUEFALSE=%KINDLETRUEFALSE% > %intermediate%\%mmdfile_edi-
tion%)
```

Then comes the building. EPUB is simple:

```
IF /I "%EXT%" == "epub" (%pandoc% --smart -o %outputfilename% --epub-stylesheet styles.c-
ss --epub-cover-image=cover-epub.jpg -f %formatextensions% --highlight-style pyg-
ments --toc --toc-depth=2 --mathjax -t epub3 %intermediate%\%mmdfile_edition%)
```

Kindle also generates an EPUB, which it then passes to kindlegen.exe:

```
IF /I "%EDITION%" == "Kindle" (%pandoc% --smart -o %outputfile-
name% --epub-stylesheet styles.css --epub-cover-image=cover-epub.jpg -f %formatexten-
sions% --highlight-style haddock --toc --toc-depth=2  -t epub3 %intermediate%\%mmdfile_ed-
ition%

%kindlegen% %outputfilename%
```

PDF (whether for online or print) is significantly more complicated. PDFs need a table of contents and an index, which have to be generated from the TEX file. But first we need the TEX file. But before that, we have to change a couple of things in the MMD file before the TEX conversion, using the custom preprocessor tool. The steps are:

- Preprocess MMD file to replace some formatting tokens for better Latex code.
- Convert MMD file to TEX file using mmd.exe
- Convert TEX file to PDF using pdflatex.
- Extract index from the PDF using makeindex.
- Re-run conversion, using the index.

```
IF /I "%EXT%" == "pdf" (copy %intermediate%\%mmdfile_edition% %intermediate%\temp.mmd
%preprocessor% %intermediate%\temp.mmd ```cs=```{[Sharp]C} ```fasm=```{[x86masm]Assem-
bler} > %intermediate%\%mmdfile_edition%
%mmd% -t latex --smart --labels --output=%intermediate%\%latexfile_edition% %intermedi-
ate%\%mmdfile_edition%
%pdflatex% %intermediate%\%latexfile_edition% > %intermediate%\pdflatex1.log
%makeindex% %intermediate%\%indexinputfile% > %intermediate%\makeindex.log
```

```
%pdflatex% %intermediate%\%latexfile_edition% %intermediate%\%indextexfile% > %intermedi-
ate%\pdflatex2.log
```

Notice the dumping of output to log files. I used those to diagnose when I got either the MMD or Latex wrong.

There is more in the batch file. The point is to hide the complexity–there are NO manual steps in creating the publishable files. Once export that MMD from Scrivener, I just type "build.bat [edition]" and away I go.

Source Control

Everything is stored in a Subversion repository which I host on my own server. I would have used Git since that's all the rage these days, but I already had this repo from the first edition. The top level of the 2nd edition directory structure looks like this:

- BetaReaders – instructions and resources for my beta readers
- Bin – Location of build.bat as well as all the tools used to build the book.
- Business – Contracts and other documents around publishing.
- Cover – Sources and Photoshop file for the cover.
- CustomToolsSrc – Source code for my preprocessor and other tools that helped me build book.
- Graphics – Visio and other source files for diagrams.
- HighPerfDotNet2ndEd.scriv — where Scrivener keeps its files (in RTF format for the content, plus a project file)
- Image – All images for the book, screenshots or exported from diagrams.
- Marketing – Some miscellaneous images I used for blog posts, Facebook stuff.
- Metadata – License text for eBook and styles.css for EPUB formatting.
- mmd – output folder for MMD file (which I also check into source control–so I can do a build without having Scrivener installed, if necessary).
- Output – Where all output of the build goes (working files go to output\intermediary)
- Resources – Documents, help files, guides, etc. that I found that were helpful, e.g., a Latex cheat sheet.
- SampleChapter – PDF of sample chapter.
- SampleCode – Source code for all the code in the book.

Tools

- Scrivener – What I manage the book's manuscript in. Has a bit of a learning curve, but once you get it set up, it's a breeze. Only thing I wish is for a command-line version that I could integrate into my build.bat.
- PdfLatex.exe – You need to get a Latex distribution. I used texlive, but there are more–probably some a lot more minimal than the one I got. It's big. This is what creates the PDF files.
- Pp.exe – an opensource preprocessor I used for a while, but it ended up not being able to handle one

situation.

- Preprocessor.exe – Custom tool I wrote to use instead of pp.exe.
- Multimarkdown – Scrivener has an old version built-in, but I ended up using the new version outside of it for some more control.
- Pandoc – I use this for the conversion from MMD to EPUB (and Word for internal use).
- Adobe Digital Editions 3.0 – This is useful for testing EPUB files for correct display. I couldn't get the latest version to work correctly.
- Calibre – In particular, the EPUB editor. For the first edition, I actually edited all the EPUB editions using this to clean up and correct the garbage Word output. For the second edition, I just used it for double-checking and debugging conversion issues.

Useful Web Sites

- Multimarkdown Cheat Sheet
- TeX on StackExchange – get all your Latex questions answered here.
- Latex on Wikibooks – Best place to get started. Most basic Latex stuff covered here.

Check out my latest book, the essential, in-depth guide to performance for all .NET developers:

Writing High-Performance.NET Code, 2nd Edition by Ben Watson. Available for pre-order:

- Amazon
- and more, see book site

This entry was posted in Books, Tips and tagged .net, book, dotnet, self-publishing, writing on July 7, 2018 [http://www.philosophicalgeek.com/2018/07/07/how-to-write-and-publish-a-technical-book-and-not-lose-your-sanity/] .