Eltanin: Phil's document processor (version 2.1)

Phil Brooke, Green Pike Ltd

11 Jan 2023

Outline of document processing

1 Write slightly-enhanced Pandoc markdown into a file in input/whatever.txt. As with Markdown generally, the intention is that the source Markdown files should be generally readable without excessive use of tags or formatting instructions (although some are inevitable). This is the main rationale for not using Docbook (XML) (although assemblies have some potential for organising a knowledge base).

2 https://garrettgman.github.io/rmarkdown/authoring_pandoc_markdown.html provides reasonable detail of Pandoc's markdown notation.

Major changes

core.5 to core.6

3 This is substantially revised from the core.5 version, which used multiple separate files. In particular, the use of Make is minimal and optional, and the main driver is (from core.6) a Python3 program.

core.19 to core.20

4 The input/output file structure was reorganised so they mirror each other. The YAML configuration was changed to move common configuration stanzas into their own list. The local variable `output_stem` was changed to `file_stem`.

Summary of directives for document authors

5 This section is aimed at those updating documents where the configuration has already been prepared.

General

6 The directives and macros are *case-sensitive*.

7 Directives are bracketed with dollar signs, i.e., $…$ (with a few exceptions). They are generally a single letter, a colon, then a key.

Setting references to sections, paragraph and other locations

Var2 VarND Var1

8 Targets for a cross-reference (“inbound xrefs”):

- p — assigns the current paragraph as target and its number as value
- s — assigns the current section as target and its number as value
- t — assigns a target (but no value)

9 `x` is a special directive that adds a prefix. This is useful when the same document source produces multiple outputs. Otherwise, multiple output files from a single input file would result in duplicate cross-reference targets.

10 `l` can be used to set local variables, not visible outside that particular document and can be repeated across documents. This is useful for building titles, and the default structure and reports expect a small number to be set. The syntax is a variable name, equals sign, and its value.

Overriding paragraph numbering occasionally

11 The `npn` directive *between exclamation marks, not dollar signs* at the *very start of a paragraph* will cause numbering of that paragraph to be omitted (completely; internal cross-referencing is also disabled for that paragraph).

Setting anchor name for a paragraph

12 Markdown provides for a section id to be set using {#…}. The prefix directive `!np!…` at the very start of a paragraph overrides the default anchor for a numbered paragraph to the remainder of that string.

Reference to values and targets

13 Referring to a cross-reference (“outbound xrefs”) and values (including those set by `l` directives):

- v — shows the value (from a p, s or l directive, error if it’s a t target)
- b — produces the URL in brackets (*parentheses*, *not* the usual delimiter)
- a — produces the URL in angle brackets
- h — produces the value (as for v) wrapped in an anchor pointing to it (as if b)
- Upper-case variants of `v`, `b`, `a` and `h`. Too many back-references from a single (part of a) document can be messy, so `V`, `B`, `A` and `H` behave as their lower-case variant but without creating a back-reference.

Longer details

14 Configuration is available via a YAML file. Quite a lot can be configured: see `settings` within the Python source. Consider using `yamllint` to validate configuration files.

15 Customisation is also available via

- macros (both simple and more complicated)
- CSS

- hacking the Python code — this is recommended when carrying out fixups, especially to the resulting HTML. One pattern is to set a local variable via the source document, then use a fixup to replace a placeholder in the Pandoc template

16 All the outputs should magically appear in the output directory.

17 The document sources and auxiliary files are stored in a Git-controlled repository. This means that tags and branches can be used to manage development versions, proposed versions and record an audit trail of changes and releases. By including the build instructions with any particular version, then it should remain possible to replicate the outputs of any commit (provided that the dependencies in section 14 continue to provide the same output).

Processor passes

18 The Python program drives a series of incremental changes to the files. These intermediate steps are normally hidden, but can be shown with the `--steps` command line option.

19 In rough order, the passes:

- Read the configuration, input files and various macros
- Apply the macros / definitions (and remove them when no longer needed)
- Modify the Pandoc AST to add section and paragraph numbers.
- Build a set of cross-references, then apply them to resolve various values and links
- Insert back-references
- Remove some “stuffing” data
- Use Pandoc to write HTML
- (optionally) use Chromium to convert HTML to PDF

Configuration structure

20 The YAML file comprises a series of keys. The most notable are

- `outputs` which describes common groups of outputs
- `configs` which describes common groups of definitions
- `short` which gives the back-reference codes for each output file

21 The `configs` block gives:

- `config` — a name to refer to the config
- `secnum` and `paranum` are boolean flags as to whether paragraph and section numbers should be shown
- `toc` is optional and default to false. If set true, include a table of contents
- `defns` — a list of the macro files to be used for these outputs

22 The `outputs` block gives:

- `dir` — the output directory — these can be shared amongst multiple output blocks

- `config` — a reference to a `config` in the `configs` block. If omitted, then each file needs an entry in `aconfs`
- `aconfs` — alternative configs. A list of output name to a particular config. Useful for overriding the configuration for a file in a particular directory
- `html` — list of HTML outputs wanted
- `pdf` — list of PDF outputs wanted — this must be a subset of the previous
- `catalogue_title` — set the name of the catalogue
- `catalogue_order` — a list of names to sort orders, where negative is at the end
- `no_catalogue` — if set `True`, inhibits generation of the catalogue file
- `map` — the relative path to other output blocks

Macros/definitions

Simple

23 These definitions are contained in a file starting with the line `simple`. Thereafter, the first word of each line is the directive to be searched for, and the rest of the line (after a single space) is the replacement text.

24 These are most useful for short imperatives and applying consistent formatting with Pandoc.

Complex

25 The first line is `complex`, followed by a series of definitions.

26 Each definition takes the name of the directive/macro, the number of arguments (0 to 10) and then its definition (in `[{…}]`).

27 These are particularly useful for

- applying variation, e.g., a macro that only provides output for some variants and not others
- inserting common blocks of text

Built-in

28 There is a single built-in macro, $*include*, which includes another file. The path is relative to the master working directory, not the including file's location.

Choice of delimiters

29 The default is `$.` `!` works reasonably well, but is less visually clear.

30 @ would be better but repeatedly conflicts with existing features, particularly citation support.

31 The macro expansions and their arguments are quoted with `[{…}]` to make collisions less likely.

Choice of macro method

32 A simplistic build of simple and slightly more complex macros was implemented directly in Python for the following reasons.

- m4 was used originally. However, it's less ideal due to the need to chain operations together and a hope of porting this to Windows (the native environment is Linux). More dependencies are problematic.
- m4's experimental (configuration option) *changeword* option could be useful
- Pandoc's Lua filters and Lua's gsub (or similar) function — but this is more challenging because of the need to scan through all Str tokens and possibly split them into at punctuation into `pandoc.Strong()` and `pandoc.Str()`.

Choice of Pandoc numbering

33 Initially, Lua filters were used. However, these are slightly harder to manipulate in general for this purpose than directly mangling the Pandoc AST via its JSON export.

34 Alternatives considered and rejected:

- CSS numbering (e.g., via `:before`). Rejected because we'd still have to count through the items to build the value-labels within the cross-referencing.
- Using m4 would mean parsing paragraphs to match Pandoc's.

Cross-references (xrefs)

35 A set of directives (see section above) can be inserted into the input files.

36 The requirement this addresses is to be able to build consistent cross-references across a range of documents. The final destinations of these documents may be across multiple directories (or even servers) despite being built from a single input directory. Additionally, being able trace backwards is valuable to see which documents (and sections or paragraphs within them) are being referred to.

37 If the capabilities of the `v` directive (to show a value rather than build a link) were not needed, then Pandoc spans and inline references may have been viable. However, these identifiers are set too late for a post-processor to handle (or at least would need a restructure); and cross-document referencing would still require some external assistance.

38 Any identifiers set for sections in Pandoc using {…} will be preserved.

39 Duplicate cross-references result in errors, as do missing targets/values.

Resolving cross-references

- `p` — removed entirely — the paragraph number span (or an empty span if visible numbering is disabled) is used as a target. Back-reference markers are also inserted for each inbound v/a/h/b reference and a link symbol.
- `s` — removed entirely — the section header is used as a target. Back-reference markers are also inserted for each inbound v/a/h/b reference along with a section sign as a link.
- `t` — replaced with an empty anchor as target (except for a link symbol). Back-reference markers are also inserted for each inbound v/a/b reference
- v — replaced with the value of the paragraph or section number, or variable (error if this points to a `t` target)
- a — *effectively* replaced with the URL in angle brackets <…>. Because pandoc can't recognise some of the URLs we use as a URL in <…>, this pass instead uses a pair `[…](…){class=a}` to obtain the desired result
- h — similar to a except the contents of `[…]` is the value. It is suffixed `[…](…){class=h}` for stying
- b — replaced with the URL in parantheses `(…)`
- `V`, `A`, `H`, `B` — same as for v/a/h/b but doesn't result in the back-reference markers

40 Note, a single space, LF, or CR-LF after `x`, `p`, `s` or `t` are also removed. This makes the original markdown source easier to read by allowing some whitespace. If whitespace is desired after one of these directives, just use (at least) two items of whitespace.

Render HTML

41 The (near) final pass generates HTML using Pandoc. CSS is applied from the default base.css file and the customisable custom.css file. A further CSS file positions and styles the paragraph numbers. The `--self-contained` option is set so that any resources such as images are embedded in the HTML file rather than referenced.

42 The default structure also includes a trailing block of HTML which uses the script `include/git-get-status` to generate a footer with Git commit information.

43 The body element is also marked with data-dir and data-file attributes to enable the use of per-directory and/or per-file styling.

Dependencies

PDP

44 Some dependencies are absolute:

- Pandoc
- Python 3

Recommended

- GNU Make
- Git (version control system)
- Chromium (for PDF output)

Copyright

PDP ET2