roam2doc parse of /home/dparker/projects/roam2doc/README.org

dparker

April 2, 2025

Contents

1 A set of python tools for convering org-roam files to a single document

- Can parse single or multiple org files, detecting and handling any roam links.
- Converts parsed org data into a tree structure, which is available for output in json format.
- Converts tree structure to html, preserving a large subset of the visual structure of the original files, and converting both internal and roam links into html links within the converted document
- Converts tree structure to latex, with with all the same features as the html version, except for differences in how document structures work in the two formats.
- Optionally converts the html to pdf using wkhtml2pdf or letex to PDF using pdflatex.
- Focused on the document creation aspect of org and org-roam, not on the todo lists, schedules, etc.
- Supports custom block surrounded with keywords `#+BEGIN_FILE_INCLUDE` ~~BEGIN_FILE_INCLUDE~ and ~#~~ `#+END_FILE_INCLUDE` where each line in the block is treated as a file include spec. The first word in the line is treated as a file path and any remainging words on the line are treated as a line of content that should be prepended to the file's content during inclusion. The path may be either full path starting at / or a path relative to the file that contains the include block. If the path resolves then whatever it contains replaces the include blog. Hopefully the contents are in org mode format, or all bets are off. The first line feature makes it possible to fit the file contents into the structure of the including file, which can be helpful in building a file that is essentially just and outline for including other files. Note that this feature would need an upgrade if you want to use it with file names that contain spaces. You can see this feature in operation when *!!! link target "file:examples/conversion/all_nodes/all_nodes.org" not found !!!* includes a couple of files in a block that looks like this (github .org file processing eats the #+BE... without the escape):

```
\#+BEGIN_FILE_INCLUDE
includer1.org ** Section heading for include file, specified in includ
includer2.org
```

```
\#+END_FILE_INCLUDE
```

- On request, it generates a PDF (only latex based option) that has special markup in the text and a cross reference at the end of the generated document that is designed to enable an AI (tested with Grok3) to figure out the internal links in the file after it has passed through OCR. This is particularly important if you are making the doc from a collection of org-roam files, as they tend to be writen as topic notes with minimal context and much of the available context arrises from the links between them. For a look at how the cross reference and annotations work you can read grok's note to future sessions in *!!! link target "file:examples/plain/note_from_grock.org" not found !!!*
- A number of example conversions are availabile in *!!! link target "file:examples/conversion" not found !!!*

2 HOW to use it

1. Until this becomes and actual package that includes an exectuable:
   (a) PYTHONPATH=. *src src* roam2doc/cli.py –help
2. There are two options for producing PDF, both of which require external tools
   (a) The preferred method is to use the latex to pdf tool "pdflatex". This is the preferred way because it generates a decent table of contents and index, and can produce special markup for AIs (using the -grokify switch). The generated latex output uses various latex packages, all of which are available on ubuntu like this:
      i. sudo apt update
      ii. sudo apt install texlive-full textlive-latex-extra
   (b) Alternatively you can generate a pdf from html using wkhtmltopdf. It currently does not produce an index or the cross reference for AIs, but it does look nicer, at least to me, so maybe it is better for humman viewing. I may eventually add an index (if I can get wkhtmltopdf to generate it) and the AI cross reference (that is just work). I can't figure

out how to build an index manually because the page breaks are created by wkhtmltopdf, and it is a fool's errand to try to put page breaks in the html. The only way to do that is with css, and even if you got it to work it would be broken any time there was an element in a page that messed up your idea of paging. Images are an example of that possibility. Or, what if the table of contents wkhtmltopdf creates takes up more paged than you expect? The whole idea is a freakin nightmare.

3. If you want to use wkhtmltopdf, then you will want to ensure that you have the patched QT version of wkhtmltopdf. The unpatched version will not handle links properly, nor produce a table of contents.

4. See some examples in action

   (a) To see the result of combining roam files:

   ```
   PYTHONPATH=./src src/roam2doc/cli.py examples/roam/roam1/roam_combi
   or
   PYTHONPATH=./src src/roam2doc/cli.py examples/roam/roam1/roam_combi
   or
   PYTHONPATH=./src src/roam2doc/cli.py examples/roam/roam1/roam_combi
   ```

   (b) To see how the include mechansim works

   ```
   PYTHONPATH=./src src/roam2doc/cli.py  examples/roam/roam2/roam_comb
   ```

   (c) To see the result of parsing a large number of org content types:

   ```
   PYTHONPATH=./src src/roam2doc/cli.py examples/conversion/all_nodes/
   ```

   (d) To see the result of parsing this file:

   ```
   PYTHONPATH=./src src/roam2doc/cli.py README.org -o readme.html --ov
   ```

   The full help:

   ```
   usage: cli.py [-h] [-o OUTPUT] [-t {html,json,latex,pdf}] [-j] [-g]

   Convert org-roam files to HTML documents.

   positional arguments:
   input                 Input file (.org), directory containing .org

   options:
   ```

```
-h, --help            show this help message and exit
-o OUTPUT, --output OUTPUT
Output file path for HTML (default: print to stdout)
-t {html,json,latex,pdf}, --doc_type {html,json,latex,pdf}
Output file path for HTML (default: html)
-j, --include_json    Include a json version of the parsed document
-g, --grokify         Produce a link cross reference table in pdf s
-l {error,warning,info,debug}, --logging {error,warning,info,debug}
Enable logging at provided level, has no effect if output goes to s
--overwrite           Allow overwriting existing output file (defau
--wk_pdf              Use wkhtmltopdf to convert output to PDF
```

3 Things to know

3.1 Things it does that might surpise you

- Org Keyword strings are stripped from the text during parsing. The only keyword that has any effect is the #+NAME: keyword, which (if at line beginning) is applied to the next non-keyword line. This allows you to name an element (e.g. a table) and then link to it by name

3.2 Things it doesn't do and probably should

- Footnotes are not parsed, they will be treated as ordinary text
- Drawers that are either property drawers at the beginning of a file or are property drawers for heaading are parsed, all other drawers are not parsed, just treated as text.
- Verbatim strings cannot contain equal sign "=", use ~ (inline code) if you need that in your text.

3.3 Things it doesn't do and maybe never will

- Parse and do something useful with the time management aspects of org files.

- Inlinetasks are not parsed, they will be treated as headings and will make things ugly

3.4 Things it doesn't do and probably won't

- Run wkhtml2pdf or pdflatex on windows. Works on linux, will probably work on Mac. You can product the html or latex output on Windows (probably, I haven't tried but it is pure python using only standard libraries. I may have gotten sloppy with file paths somewhere, but maybe not).

3.5 Things that might be nice to add someday

- Produce output including any LaTex features found in the org files
- Provide option to allow uset to supply css and or javascript contents to be merged into the head of the html output. There is already an option to include a json object version of the parsed tree into the head, so you could write code to inspect that object and do interesting things. Of course you can do this just by editing the output directly.

3.6 History, what I wanted and why it lead to this.

3.6.1 What for?

I wanted to be able to take notes on a wide range of topics and relate them together into a book outline. Orgroam perfectly fit my style, so I started learning it.

3.6.2 First problem

I had also just started using the Grok3 AI to work on the research I was turning into notes, so I wanted to be able to load all the notes into the Grok context before submitting prompts. Grok informed me that orgroam files would not work as well as I wanted because it wouldn't do well interpreting the org files, and especially the links. Grok suggested that I would get much better results if I could collect the files into single document such as a PDF. So I needed a tool to do this. I prefer to look for python based solutions to such problems since I can modify or extend them if I need to, python being my favorite language.

3.6.3 The First Fix

I found the pyorg package at https:*/github.com*nasa9084/py-org . Its main purpose was to export org content to html, and I have experience using wkhtmk2pdf to cre-

ate PDFs, so that seemed workable. I forked to https:/*github.com*dlparker/pyorg2 and was able to quickly modify it to add support for roam links. I got Grok to help me by updating the tests from nodetest to pytest. I then upgraded the tests to get 100% coverage. Seemed like a good start

3.6.4 Now I have two problems

As I started looking at how I wanted to use this, it became clear that I also wanted to support org internal links, which the orignal package did not. The linking to something part is simple, but the range of link targets that org supports lead to some complexity when thinking about adding it to the package. For example, you can link to a Table and almost any other element of an org file but giving it a name using a #NAME+ keyword like so:

```
#+NAME: my_table
| col 1         | col 2            |
| row 1 col 1 | row 1 col 2      |

[[my_table][link to my table]]
```

Also adding complexity to the needed changes is the fact that a link/reference can appear in many places other than plain text. Inside table cells, for example.
The original package's parsing had some other limitations as well, which may well have been the author's intention to keep the task at hand to a useful limited subset of org format. The full format is pretty rich. See https:/*orgmode.org*worg/org-syntax.html

The scale of the modifications needed to achieve my goals convinced me that I was going to contort the structure so badly that it would be dificult to maintain. So I decided to start over.

Index