TP-NOTE(1) Version 1.23.3 | Tp-Note documentation

manpage

Jens Getreu

Table of Contents

1. NAME

Tp-Note - save and edit your clipboard content as a note file.

2. SYNOPSIS

```
tpnote [-a ] [-b] [-c <FILE>] [-C <FILE>] [-d <LEVEL>] [-e] [-l <LANG>]
       [-p <NUM>] [-n] [-t] [-u] [-v] [-V] [-x <DIR>|''|'-']
       [<DIR>|<FILE>]
```

3. DESCRIPTION

Tp-Note is a note-taking tool and a template system, that synchronizes the note's metadata with its filename. Tp-Note analyses its environment and the clipboard content and stores the result in variables. New notes are created by filling these variables in predefined and customizable *Tera*-templates. In case the first positional parameter "`<FILE>`"points to an existing Tp-Note file, the note's metadata is parsed and, if necessary, its filename is adjusted. For all other file types, Tp-Note creates a new note in the same directory annotating the file. If the positional parameter "`<DIR>`"points to an existing directory (or, when omitted, the current working directory), a new note is created in that directory. After creation, Tp-Note launches the systems file editor. Although the configurable default templates are written for Markdown, Tp-Note is not tied to any specific markup language. However, Tp-Note comes with an optional viewer feature, that currently renders only Markdown, ReStructuredText and HTML input. In addition, there is some limited support for Asciidoc and WikiText. Finally, the note's rendition is live updated and displayed in the user's web browser.

After the user finished editing, Tp-Note analyses potential changes in the notes metadata and renames, if necessary, the file, so that its metadata and filename are in sync again. Finally, the resulting path is printed to "`stdout`", log and error messages are dumped to "`stderr`".

This document is Tp-Note's technical reference. More information can be found in Tp-Note's user manual[1] and at Tp-Note's project page[2].

4. OPERATION MODES

Tp-Note operates in 5 different modes, depending on its command line arguments and the clipboard state. Each mode is associated with one content template and one filename template.

4.1. Create a new note with empty clipboard

In case the clipboard is empty while starting, the new note is created with the templates: "`tmpl.from_dir_content`" and "`tmpl.from_dir_filename`". By default, the new note's title is the parent's directory name. The newly created file is then opened with an external text editor, allowing it to change the proposed title and add other content. When the text editor closes, Tp-Note synchronizes the note's metadata and its filename. This operation is performed with the "`tmpl.sync_filename`" template.

Example: the clipboard is empty and "`<path>`" is a directory (or empty):

```
tpnote "./03-Favorite Readings/"
```

or

```
cd "./03-Favorite Readings"
tpnote
```

creates the document:

```
./03-Favorite Readings/20211031-Favorite Readings--Note.md
```

with the content:

[1] https://blog.getreu.net/projects/tp-note/tpnote--manual.html
[2] https://blog.getreu.net/projects/tp-note/

```
---
title:      Favorite Readings
subtitle:   Note
author:     Getreu
date:       2021-10-31
lang:       en-GB
---
```

4.2. Create a new note based on clipboard data

When "`<path>`" is a directory and the clipboard is not empty, the clipboard's content is stored in the variable "`{{ clipboard }}`". In addition, if the content contains an hyperlink in Markdown format, the hyperlink's name can be accessed with "`{{ clipboard | link_text }}`", its URL with "`{{ clipboard | link_dest }}`" and its title with "`{{ clipboard | link_title }}`". The new note is then created with the "`tmpl.from_clipboard_content`" and the "`tmpl.from_clipboard_filename`" templates. Finally, the newly created note file is opened again with some external text editor. When the user closes the text editor, Tp-Note synchronizes the note's metadata and its filename with the template "`tmpl.sync_filename`".

Note: this operation mode also empties the clipboard (configurable feature).

Clipboard simulation

When no mouse and clipboard is available, the clipboard feature can be simulated by feeding the clipboard data into `stdin`:

```
echo "[The Rust Book](<https://doc.rust-lang.org/book/>)" | tpnote
```

Tp-Note behaves here as if the clipboard contained the string: "`[The Rust Book](<https://doc.rust-lang.org/book/>)`".

The clipboard contains a string

Example: While launching Tp-Note the clipboard contains the string: "`Who Moved My Cheese?\n\nChapter 2`" and "`<path>`" is a directory.

```
tpnote "./03-Favorite Readings/"
```

or

```
cd "./03-Favorite Readings/"
tpnote
```

This creates the document:

```
./03-Favorite Readings/20211031-Who Moved My Cheese--Note.md
```

with the content:

```
---
title:      Who Moved My Cheese
subtitle:   Note
author:     Getreu
date:       2021-10-31
lang:       en-GB
---

Who Moved My Cheese?

Chapter 2
```

We see from the above example, how the “`tmpl.from_clipboard_content`”content template extracts the first line of the clipboards content and inserts it into the header’s “`title:`”field. Then, it copies the entire clipboard content into the body of the document. However, if desired or necessary, it is possible to modify all templates in Tp-Note’s configuration file. Note, that not only the note’s content is created with a template, but also its filename: The “`tmpl.from_clipboard_filename`”filename template concatenates the current date, the note’s title and subtitle.

The clipboard contains a hyperlink

Example: “`<path>`”is a directory, the clipboard is not empty and it contains the string: “`I recommend:\n[The Rust Book](https://doc.rust-lang.org/book/)`”.

```
tpnote './doc/Lecture 1'
```

Tp-Note’s templates “`tmpl.from_clipboard_content`” and “`tmpl.from_clipboard_filename`” create the following document:

```
./doc/Lecture 1/20211031-The Rust Book--Notes.md
```

```
---
title:      The Rust Book
subtitle:   URL
author:     Getreu
date:       2021-10-31
lang:       en-GB
---

I recommend:
[The Rust Book](<https://doc.rust-lang.org/book/>)
```

When analyzing the clipboard's content, Tp-Note searches for hyperlinks in Markdown, ReStructuredText, Asciidoc and HTML format. When successful, the content template uses the link text of the first hyperlink found as document title.

The clipboard contains a string with a YAML header

Example: "`<path>`"is a directory, the clipboard is not empty and contains the string: "`---\ntitle: Todo\nfile_ext: mdtxt\n---\nnothing`".

```
tpnote
```

This creates the note: "`20230915-Todo.mdtxt`" with the following content:

```
---
title:      Todo
subtitle:   Note
author:     Getreu
date:       2023-09-15
lang:       fr-FR

file_ext:   mdtxt
---

nothing
```

Technically, the creation of the new note is performed using the YAML header variables: "`{{ fm_title }}`", "`{{ fm_subtitle }}`",

“`{{ fm_author }}`”, “`{{ fm_date }}`”, “`{{ fm_lang }}`”, “`{{ fm_sort_tag }}`” and “`{{ fm_file_ext }}`” which are evaluated with the “`tmpl.from_clipboard_yaml_content`” and the “`tmpl.from_clipboard_yaml_filename`” templates.

Note, that the same result can also be achieved without clipboard input by typing in a terminal:

```
echo -e "---\ntitle: Todo\nfile_ext: mdtxt\n---\n\nnothing" | tpnote
```

Furthermore, this operation mode is very handy with pipes in general, as shows the following example: it downloads some webpage, converts it to Markdown and copies the result into a Tp-Note file. The procedure preserves the webpage’s title in the note’s title:

```
curl 'https://blog.getreu.net' \
| pandoc --standalone -f html -t markdown_strict+yaml_metadata_block \
| tpnote
```

creates the note file “`20230919-Jens Getreu's blog--Note.md`” with the webpage’s content converted to Markdown:

```
---
title:      Jens Getreu's blog
subtitle:   Note
author:     Getreu
date:       2023-09-15
lang:       en

viewport:   width=device-width, initial-scale=1.0, maximum-scale=1
---

<a href="/" class="logo">Jens Getreu's blog</a>

-   [Home](https://blog.getreu.net)
-   [Categories](https://blog.getreu.net/categories)
```

4.3. Create a new note annotating a non Tp-Note file

When “`<path>`” points to an existing file, whose file extension is other than “`.md`”, a new note is created with a similar filename and a reference to the original file is copied into the new note’s body. If the clipboard contains some text, it is appended there also.

The logic of this is implemented in the templates: “`tmpl.annotate_file_content`” and “`tmpl.annotate_file_filename`”. Once the file is created, it is opened with an external text editor. After editing the file, it will be - if necessary - renamed to be in sync with the note’s metadata.

Example:

```
:> "Classic Shell Scripting.pdf"

tpnote "Classic Shell Scripting.pdf"
```

creates the note:

```
Classic Shell Scripting.pdf--Note.md"
```

with the content:

```
---
title:      Classic Shell Scripting.pdf
subtitle:   Note
author:     Getreu
date:       2023-09-15
lang:       en-US
---

[Classic Shell Scripting.pdf](<Classic Shell Scripting.pdf>)
```

The configuration file variable “`filename.extensions`”list all the file extensions that Tp-Note recognizes as own file types. Only foreign file types can be annotated.

Note that the file annotation mode also reads the clipboard’s content: when it is not empty, its data is appended to the new note’s body.

4.4. Convert a text file into a Tp-Note file

Consider the content of the following text file “`Ascii-Hangman--A game for children.md`” whose creation date is 13 March 2022:

```
A little game designed for primary kids to revise vocabulary in classroom.
```

To convert the text file into a Tp-Note file type:

```
tpnote --add-header --batch "Ascii-Hangman--A game for children.md"
```

NB: the "`--add-header`"flag is actually not necessary, as it is enabled by default through the configuration file variable "`arg_default.add_header = true`".

As a result of the above command, Tp-Note converts the filename into:

```
20220313-Ascii-Hangman--A game for children.md
```

and prepends a YAML header to the file's content:

```
---
title:      Ascii-Hangman
subtitle:   A game for children
author:     Getreu
date:       2022-03-13
lang:       en-US

orig_name:  Ascii-Hangman--A game for children.md
---

A little game designed for primary kids to revise vocabulary in classroom.
```

4.5. Use Tp-Note in shell scripts

- **Use case: download a webpage and store it as Tp-Note file**

  Using the method displayed above you can save time and create a script with:

  ```
  sudo nano /usr/local/bin/download
  ```

  Insert the following content:

  ```
  #!/bin/sh
  ```

```
curl "$1" | pandoc --standalone -f html -t markdown_strict
+yaml_metadata_block | tpnote
```

and make it executable:

```
sudo chmod a+x /usr/local/bin/download
```

To execute the script type:

```
download 'https://blog.getreu.net'
```

- **Use case: synchronize recursively filenames and metadata**

  The following synchronizes bidirectionally all filenames with the note's YAML header data.

```
TPNOTE_USER="John" find . -type f -name '*.md' -exec tpnote -a -b {}
 > /dev/null \;
```

  The direction of the synchronization depends on whether the "`.md`"file has a valid YAML header or not:

  # A YAML header is present and valid: the header fields might update the filename (see template "`tmpl.sync_filename`"). A possible *sort-tag* at the beginning of the filename remains untouched.

  # No YAML header: a new header is prepended (see template "`from_text_file_content`") and the filename might change slightly (see template "`from_text_file_filename`"). A possible *sort-tag* at the beginning of the filename remains untouched. If the filename does not start with a sort tag, the file's creation date is prepended.

4.6. Editing notes

Unless invoked with "`--batch`"or "`--view`", Tp-Note launches an external text editor after creating a new note. This also happens when "`<path>`" points to an existing "`.md`"-file.

Example: edit the note from the previous example:

```
cd "./03-Favorite Readings"
```

```
tpnote 20211031-Favorite Readings--Note.md
```

4.7. Automatic filename synchronization before and after editing

Before launching the text editor and after closing it, Tp-Note synchronizes the filename with the note's metadata. When the user changes the metadata of a note, Tp-Note will replicate that change in the note's filename. As a result, *all your note's filenames always correspond to their metadata*, which helps to retrieve your notes in large data pools.

Example:

```
tpnote "20200306-Favorite Readings--Note.md"
```

The way how Tp-Note synchronizes the note's metadata and filename is defined in the template "`tmpl.sync_filename`".

Once Tp-Note opens the file in your text editor, let's assume you decide to change the title in the note's YAML metadata section from "`title: Favorite Readings`"to "`title: Introduction to bookkeeping`". After closing the text editor, Tp-Note updates the filename automatically:

```
20200306-Introduction to bookkeeping--Note.md
```

Note: the sort tag "`20200306`"has not changed. The filename synchronization mechanism by default never does. (See below for more details about filename synchronization).

5. OPTIONS

-a, **--add-header**

> Prepends a YAML header in case the text file does not have one. The default template, deduces the "`title:`" and "`subtitle:`" header field from the filename. It's sort-tag and file extension remain untouched. In case the filename is lacking a *sort-tag*, the file creation date in numerical format is prepended. As this option is activated by default, it has no effect unless you set "`arg_default.add_header = false`" in the configuration file.

-b, **--batch**

Do not launch the external text editor or viewer. All other operations are available and are executed in the same way. In batch mode, error messages are dumped on the console only and no alert windows pop up.

Tp-Note ignores the clipboard when run in batch mode with "`--batch`". Instead, if available, it reads the `stdin` stream as if the data came from the clipboard.

-c *FILE*, **--config**=*FILE*

Loads an additional configuration from the TOML formatted *FILE* and merges it into the default configuration.

-c *FILE*, **--config-defaults**=*FILE*

Dumps the internal default configuration in TOML format into *FILE*.

-d *LEVEL*, **--debug**=*LEVEL*

Prints additional log messages. The debug level *LEVEL* must be one out of "`trace`", "`debug`", "`info`", "`warn`", "`error`" (default) or "`off`". The level "`trace`" reports the most detailed information, while "`error`" informs you only about failures. A "`warn`" level message means, that not all functionality might be available or work as expected.

Use "`-b -d trace`" for debugging templates and "`-V -b -d trace`" for debugging configuration files. If the HTTP server (viewer) does not work as expected: "`-n -d debug`". If your text editor does not open as expected: "`-n -d info --edit`". Or, to observe the launch of the web browser: "`-n -d info --view`". The option "`-d trace`" shows all available template variables, the templates used and the rendered result of the substitution. This is particularly useful for debugging new templates. The option "`-d off`" silences all error message reporting and also suppresses the error pop-up windows.

Note, under Linux, when `-d trace` is given, no pop-up messages appear. Instead, the logs are dumped to the console from where you started Tp-Note.

All error messages are dumped in the error stream `stderr` and appear on the console from where Tp-Note was launched:

```
tpnote.exe --debug info my_note.md
```

Under Windows the output must be redirected into a file to see it:

```
tpnote.exe --debug info my_note.md >debug.md 2>&1
```

Alternatively, you can redirect all log file entries into popup alert windows.

```
tpnote.exe --popup --debug info my_note.md
```

The same can be achieved by setting following configuration file variables (especially useful with Windows):

```
[arg_default]
debug = 'info'
popup = true
```

The value for "`arg_default.debug`" must be one out of "`trace`", "`debug`", "`info`", "`warn`", "`error`"(default) and "`off`". They have the same meaning as the corresponding command line options.

-e, **--edit**

Edit only mode: opens the external text editor, but not the file viewer. This disables Tp-Note's internal file watcher and web server, unless "`-v`"is given. Alternatively you can set the environment variable "`TPNOTE_BROWSER=""`" to the empty string. Another way to permanently disable the web server is to set the configuration variable "`arg_default.edit=true`". When "`--edit --view`" appear together, both the editor and the viewer will open and the `arg_default.edit` variable is ignored.

-l *LANGUAGE_TAG*, **--force-lang**=*LANGUAGE_TAG*

Disables the automatic language detection while creating a new note file and use *LANGUAGE_TAG* instead. *LANGUAGE_TAG* is formatted as IETF BCP 47 language tag, e.g. "`en-US`". If *LANGUAGE_TAG* equals "`` `', the environment variable 'TPNOTE_LANG` `` determines the language instead; or, if the latter is not defined, the user's default language, as reported from the operating system's locale setting, is decisive.

-p *PORT*, **--port**=*PORT*

Sets the server port that the web browser connects to, to the specified value *PORT*. If not given, a random available port is chosen automatically.

-n, **--no-filename-sync**

Whenever Tp-Note opens a note file, it synchronizes its YAML-metadata with its filename. "`--no-filename-sync`" disables this synchronization. In addition, in scripts this flag can be especially useful for validating the syntax of "`.md`"-files. See section EXIT STATUS for more details. The section METADATA FILENAME SYNCHRONIZATION shows alternative ways to disable synchronization.

-s *PORT*, **--scheme**=*SCHEME_NAME*

Sets the filename scheme for creating a new note file. This overwrites the "`arg_default.scheme`" value in the configuration file. Under "`[[scheme]]`" follows the definition of the schemes. The default configuration ships two schemes with the SCHEME_NAMES "`default`"and "`zettel`" (for Zettelkasten).

-t, **--tty**

Tp-Note tries different heuristics to detect whether a graphic environment is available or not. For example, under Linux, the "`DISPLAY`" environment variable is evaluated. The "`--tty`"flag disables the automatic detection and sets Tp-Note into "console only" mode: now only the non GUI editor (see configuration variable: "`app_args.editor_console`") and no viewer is launched.

-u, **--popup**

Redirects log file entries into popup alert windows. Must be used together with the **--debug** option to have an effect. Note, that debug level "`error`"conditions will always trigger popup messages, regardless of **--popup** and **--debug** (unless "`--debug off`"). Popup alert windows are queued and will never interrupt Tp-Note. To better associate a particular action with its log events, read through all upcoming popup alert windows until they fail to appear.

-v, **--view**

View only mode: do not open the external text editor. This flag instructs Tp-Note to start an internal file watcher and web server and connect the system's default web browser to view the note file and to observe live file modifications. The configuration setting "`arg_default.edit=true`"or the environment variable "`TPNOTE_EDITOR=""`" disables the viewer. However, with "`--`

`view`" given at the command line, the viewer appears, regardless of the value of "`arg_default.edit`".

NB: By default, Tp-Note tries to synchronize every file it opens. To prevent the viewed filename from changing, "`--view`"can be used together with "`--no-filename-sync`".

-V, **--version**

Print Tp-Note's version, its built-in features and the path to the sourced configuration file. The output is YAML formatted for further automatic processing. In addition, use "`-V -b -d trace`" for configuration file debugging.

-x *DIRECTORY*, **--export**=*DIRECTORY*

Prints the note as HTML rendition into *DIRECTORY*. "`-x -`"prints to *stdout*. The empty string, e.g. "`--export=`"or "`-x ""`", defaults to the directory where the note file resides. No external text editor or viewer is launched. Can be combined with "`--batch`" to avoid popup error alert windows.

--export-link-rewriting=*MODE*

Chooses how local links in the exported HTML file are written out: "`off`", "`short`" or "`long`" (default). No link rewriting occurs, for the *MODE* "`off`". The *MODE* "`short`" rewrites all local relative links to absolute links, whose base is the first parent directory containing the marker file "`.tpnote.toml`". NB, the directory of the marker file defines the base for all absolute local links in your Tp-Note file! The mode "`long`"rewrites *all* local links to absolute links whose base is the system's root directory "`/`". For relative local links this is performed by prepending the path to the note file. Absolute local links get the path to the marker file "`.tpnote.toml`"prepended. In case you do not place a "`.tpnote.toml`"file in a parent directory, the base for absolute local links in your note file is interpreted as "`/`".

The right *MODE* choice depends on how you view the resulting HTML: if you publish on a web server, then "`short`"might be a good choice. Do not forget to place a marker file "`.tpnote.toml`"somewhere in the document's path. If you view the HTML file directly in your web browser, better choose "`long`". In this case, the present of a marker file will not affect the output. NB: You can also set this option via Tp-Note's configuration file with the key "`arg_default.export_link_rewriting`".

6. THE NOTE'S DOCUMENT STRUCTURE

Tp-Note considers a text file to be a valid note file, if its:

- file extension is listed in one of the configuration file variables "`filename.extension_*`"; if its
- content has a valid YAML header and
- the YAML header contains a key whose name is defined in the configuration file variable "`tmpl.compulsory_header_field`" (default "`title`").

A Tp-Note note file is always UTF-8 encoded. As newline, either the Unix standard "`\n`"or the Windows standard "`\r\n`"is accepted. Tp-Note writes out newlines according the operating system it runs on.

6.1. The document's header and body

Tp-Note is designed to be compatible with "`Pandoc`'s and"`RMarkdown`s document structure as shown in the figure below. In this documentation the terms "YAML header", "header" and "front matter" are used as synonyms to designate to document's metadata block at the beginning of the text file:

```
---
<YAML-front-matter>
---

<document-body>
```

The YAML front-matter starts at the beginning of the document with "`---`" and ends with "`...`" or "`---`". Note that according to the YAML standard, string literals are always encoded as JSON strings. By convention, a valid Tp-Note file has at least one YAML field named "`title:`" (the name of this compulsory field is defined by the "`tmpl.compulsory_header_field`" variable in the configuration file and can be changed there).

Note that prepended text, placed before the YAML front-matter, is ignored. There are however certain restrictions: If present, the skipped text should not be too long (cf. constant "`BEFORE_HEADER_MAX_IGNORED_CHARS`"in the source code of Tp-Note) and it must be followed by at least one blank line:

```
Prepended text is ignored.

---
<YAML-front-matter>
---

<document-body>
```

There is no restriction about the markup language being used in the note's text body. However, the default templates assume Markdown and the file extension "`.md`". Both can be changed easily by adapting Tp-Note's configuration file. Besides the requirements concerning its header, a valid Tp-Note file must have a filename extension that is listed in one of the configuration file variables: "`filename.extension_*`". The latter also determine which internal markup language render is called for Tp-Note's internal viewer.

6.2. Links to resources and other documents

Link types

The document's body often contains (inline) links to resources e.g. images and links to other documents. This section describes how the automatic path rewriting of local links works.

In general, the link syntax depends on the markup language used in the Tp-Note file. The following examples illustrate the different link types Tp-Note understands:

Link type	**Example in Markdown notation**
Absolute URL	"`[blog](<https://blog.getreu.net>)`"
Relative URL (=local link)	"`![Alt text](<images/my logo.png>)`"
Absolute local link	"`![Alt text](</images/my logo.png>)`"
Relative local link	"`![Alt text](<images/my logo.png>)`"
Relative local link	"`[my doc](<../../notes/31-my doc.md>)`"
Relative local autolink	"`<tpnote:../../notes/31-my%20doc.md>`"
Shorthand link	"`[my doc](<../../notes/31>)`"
Shorthand autolink	"`<tpnote:../../notes/31>`"
Formatted shorthand link	"`<tpnote:../../notes/31?-->)`"

Remarks:

- The base for absolute local links is the first parent directory containing the marker file "`.tpnote.toml`". If absent, absolute local links refer to the root directory "`/`".

- *Shorthand link*: Instead of writing out the full link destination, e.g. “`[my doc](<./docs/20230508-my note.md>)`”, you can shorten the link to “`[my doc](<docs/20230508>)`” indicating only the destination’s sort-tag. Alternatively, the same shorthand link can be expressed as autolink as well: “`<http:docs/20230508>`”. NB, if more than one document with the same sort-tag exist in a directory, the viewer only displays the first in alphabetical order. To set up a different order, you can extend the sort-tag until it becomes unique, e.g. by renaming the destination document in the above example to “`./ docs/20230508a-my note.md`”. This way you obtain the unique sort-tag “`20230508a`”.

Although Tp-Note’s built in viewer follows absolute and relative local links, usually the latter are preferred. They make moving documents easier, as relative links do not break when the source and the destination documents are moved together.

As mentioned above, the shortest way to refer to other Tp-Note documents, is indicating their sort-tag only, e.g. “`<tpnote:dir/123>.`”and “`[my file](<tpnote:dir/123>)`”. If the other document is located in the same directory, the links are even shorter: “`<tpnote:123>.`” and “`[my file](<tpnote:123>)`”.

Local links in HTML export

Tp-Note’s exporter function “`--export`”converts a given Tp-Note file into HTML and adds “`.html`”to the output filename. Links in the documents content to other Tp-Note files are hereby rewritten by appending “`.html`”to their URLs. This way you can convert groups of documents to HTML and later jump from document to document in your web browser. The option “`--export-link-rewriting`”allows you to fine-tune how local links are written out. Valid values are: “`off`”, “`short`” and “`long`”.

In order to achieve this, the user must respect the following convention concerning absolute paths in local links in Tp-Note documents: When a document contains a local link with an absolute path, the base of this path is considered to be the directory where the marker file “`.tpnote.toml`” resides (or “`/`” in non exists). The option “`--export-link-rewriting`” decides how local links in the Tp-Note document are converted when the HTML is generated. If its value is “`short`”, then local links with relative paths are converted to absolute paths. The base of the resulting path is where the “`.tpnote.toml`”file resides (or `/` if none exists). Consider the following example “`--export-link-rewriting=short`”:

- The Tp-Note file “`/my/docs/car/bill.md`” contains
- a local link with an absolute path: “`/car/scan.jpg`”,
- and another link with a relative path: “`./photo.jpg`”.

- The document root marker is: “`/my/docs/.tpnote.toml`”.

The images in the resulting HTML will appear as

- “`/car/scan.jpg`”.
- “`/car/photo.jpg`”.

For “`--export-link-rewriting=short`”, in addition to the above, all absolute paths in local links are rebased to “`/`”’. Consider the following example:

- The Tp-Note file “`/my/docs/car/bill.md`” contains
- a link with an absolute path: “`/car/scan.jpg`”,
- and another link with a relative path: “`./photo.jpg`”.
- The document root marker is: “`/my/docs/.tpnote.toml`”.

The images in the resulting HTML will appear as

- “`/my/docs/car/scan.jpg`”.
- “`/my/docs/car/photo.jpg`”.

Summary: The right “`--export-link-rewriting`”choice depends on how you view the resulting HTML: if you publish on a web server, then “`short`”might be a good choice (do not forget to place a marker file “`.tpnote.toml`”somewhere in the document’s path). If you view the HTML file directly in your web browser, better choose “`long`”. In this case, the present of a marker file will not affect the output.

Local links with format strings

So far, we have seen how Tp-Note’s viewer and HTML exporter converts the *destination* of local links “`[text](destination)`”. Concerning the local link’s *text* property, the situation is simpler as the *text* property never changes during the above discussed rewriting process. However, it is possible to overwrite the displayed text property by appending a *format string* to the destination: “`[formatted destination](destination?format string)`”.

All local links in the following tables have the same link destination “`dir/01ac-Tulips--red, yellow.md`”. The examples differ only in the way the link is displayed in the browser.

Local link	**What you see**
“`[matters](<dir/01ac-Tulips--red, yellow.md>)`”	matters
“`[matters](<dir/01ac>)`”	matters

Formatted local link	What you see
“ `[whatever](<dir/01ac-Tulips--red, yellow.md?>)`”	Tulips–red, yellow
“ `[whatever](<dir/01ac?>)`”	Tulips–red, yellow
“ `[whatever](<dir/01ac??>)`”	01ac
“ `[whatever](<dir/01ac?,>)`”	Tulips–red
“ `[whatever](<dir/01ac?-->)`”	Tulips

Local autolink	What you see
“ `'<tpnote:dir/01ac-Tulips--red, %20yellow.md>`”	dir/01ac-Tulips–red,%20yellow.md
“ `'<tpnote:dir/01ac>`”	dir/01ac

Formatted local autolink	What you see
“ `'<tpnote:dir/01ac-Tulips--red, %20yellow.md?>`”	Tulips–red, yellow
“ `'<tpnote:dir/01ac?>`”	Tulips–red, yellow
“ `'<tpnote:dir/01ac?:>`”	01ac-Tulips–red, yellow.md
“ `'<tpnote:dir/01ac?:.>`”	01ac-Tulips–red, yellow
“ `'<tpnote:dir/01ac?-:,>`”	Tulips–red
“ `'<tpnote:dir/01ac?--:,>`”	red

7. METADATA FILENAME SYNCHRONIZATION

Consider the following Tp-Note file:

```
20151208-Make this world a better place--Suggestions.md
```

The filename has 4 parts:

```
{{ fm_sort_tag }}-{{ fm_title }}--{{ fm_subtitle }}.{{ fm_file_ext }}
```

The “ `-`”between “ `{{ fm_sort_tag }}`”and “ `{{ fm_title }}`”is hereafter referred to as *sort-tag separator* (cf. “ `filename.sort_tag.separator`”).

A so-called *sort tag* is an alphanumerical prefix at the beginning of the filename. It is used to order files and notes in the file system. Besides numerical digits and lowercase letters, a *sort tag* may contain any combination of "`_`", "`-`", "`=`" and "`.`" (cf. "`filename.sort_tag.extra_chars`"). If a sort-tag contains lowercase letters, only 2 in a row are allowed (cf. "`filename.sort_tag.letters_in_succession_max`"). Examples:

- *Chronological sort tag*

  ```
  20140211-Reminder.doc
  20151208-Manual.pdf
  2015-12-08-Manual.pdf
  ```

  NB: All chronological sort-tags must have at least one counter with 4 digits or more, e.g. "`2015`". The character "`-`" between the counters is optional.

  Tip: Always include the year with 4 digits in chronological sort-tags.

- *Sequence number sort tag*

  ```
  02-Invoices/
  08-Tax documents/
  09_2_144-Manual.pdf
  09.9.1-Notes.md
  ```

  NB: None of the counters exceeds 3 digits (cf. "`filename.sort_tag.chronological.digits_in_succession_min`") which is the criterium recognize a sequential sort-tag. The largest counter in this example is "`144`".

- *Alphanumerical sequence number sort tag*

  ```
  02-Invoices/
  08-Tax documents/
  09b144-Manual.pdf
  09i1-Notes.md
  ```

  NB: the example is equivalent to the previous one. The only difference is, that the separators are expressed through the alternation of digits and letters.

Summary:

1. A *sort-tag* is composed of a number of counters, which can be numerical, e.g. “`123.28`” or combined numerical/letter based, e.g. “`123ab`”.
2. A counter is set of digits (base 10) “`123`” or a set of lowercase letters (base 26) “`ab`”.
3. A letter based counter can be maximal 2 letters wide. Its maximum is “`zz`” (cf. “`filename.sort_tag.letters_in_succession_max`”).
4. A *sequential sort-tag* is a sort-tag that whose counters are at most 3 digits wide (cf. “`sort_tag.sequential.digits_in_succession_max`”).
5. The filter “`incr_sort_tag`” increments only sequential sort-tags.
6. In order not to confuse sequential and chronological sort-tags, it is recommended to always write out the year in chronological sort-tags with 4 digits, e.g. ’“`2013-08-10`” or “`20130810`”.

Before Tp-Note creates a new note file, it searches the current directory for the latest existing Tp-Note file. If that file starts with a sequence number sort-tag, Tp-Note increments that number and uses the result as sort-tag for the new note file. Otherwise, the new note gets a chronological sort tag of today.

A note’s filename is said to be in sync with its metadata, when the following holds (slightly simplified, see “`tmpl.sync_filename`”):

> filename on disk without *sort tag* == “`{{ fm_title }}--{{ fm_subtitle }}.md`”

[3]

Example, consider the following document with the filename:

```
20211031-My file.md
```

and the content:

```
---
title:      1. The Beginning
subtitle:   Note
author:     Getreu
```

[3] The variables “`{{ fm_title }}`” and “`{{ fm_subtitle }}`” reflect the values in the note’s front matter.

```
date:       2021-10-31
lang:       en-GB

remainder:  false
---
```

As “`My file.md`”is not equal to “`1. The Beginning--Note.md`”, Tp-Note will rename the file to “`20211031-1. The Beginning--Note.md`”. If the filename had been “`05_02-My file.md`”, it would rename it to “`05_02-1. The Beginning--Note.md`”.

Note: When the YAML front-matter does not contain the optional “`sort_tag`”variable, Tp-Note will never change a sort tag. Nevertheless, it might change the rest of the filename!

The reason why by default Tp-Note does not change sort tags is, that they define their order in the file listing. In general this order is independent of the notes content. The simplest way to organize the sort tags of your files is by renaming them directly in your file system. Nevertheless, in some cases you might want to have full control over the whole filename through the note’s YAML front-matter. For example, if — for some reason — you have changed the document’s date in the front-matter and you want to change the chronological sort tag in one go. In order to overwrite the note’s sort tag on disk, you can add a “`sort_tag`”string-variable to its front-matter:

```
---
title:      1. The Beginning
date:       2021-10-31

sort_tag:   '20211101'
---
```

Note, the above sort-tag value - here a number - must be enclosed with quotes in order label it as string type. When Tp-Note synchronizes the note’s metadata with its filename, it will also change the sort tag from “`20211031`”to “`20211101`”. The resulting filename becomes “`20211101-1. The Beginning--Note.md`”.

The “`sort_tag`”variable also becomes handy, when you want to create one single note without any sort tag:

```
---
title:      1. The Beginning
```

```
sort_tag:   ''
---
```

In the same way, how it is possible to pin the sort tag of the note from within the note's metadata, you can also change the file extension by adding the optional "`file_ext`"variable into the note's front-matter:

```
---
title:      1. The Beginning

file_ext:   rst
---
```

This will change the file extension from "`.md`" to "`.rst`. The resulting filename becomes"`20211101-1. The Beginning--Note.rst`'.

Important: "`rst`" must be one of the registered file extensions listed in the "`filename.extensions`"variable in Tp-Note's configuration file. If needed you can add more extensions there. If the new filename extension is not listed in one of these variables, Tp-Note will not be able to recognize the note file as such and will not open it in the external text editor and viewer.

Note: When a "`sort_tag`"variable is defined in the note's YAML header, you should not change the sort tag string in the note's file name manually by renaming the file, as your change will be overwritten next time you open the note with Tp-Note. However, you can switch back to Tp-Note's default behaviour any time by deleting the "`sort_tag`"line in the note's metadata. The same applies to the "`file_ext`" variable.

The metadata filename synchronization feature can be disabled permanently by setting the configuration file variable "`arg_default.no_filename_sync = true`". To disable this feature for one time only, invoke Tp-Note with "`--no-filename-sync`". To exclude a particular note from filename synchronization, add the YAML header field "`filename_sync: false`".

```
---
title:         1. The Beginning

filename_sync: false
---
```

Note, that in the above described examples, the information flow always goes from the YAML note header towards the note's filename. However, when Tp-Note opens a text file without a YAML header, a new header is added automatically. In this case the information flow goes from the filename towards the header, namely in the opposite direction. Once the new header is prepended to the text file, a regular filename synchronization - as described above - is triggered and executed as described above.

Technically, all rules and logic of how the synchronization is executed, are encoded in customizable so-called filename templates (cf. section *Templates*).

8. CUSTOMIZATION

Tp-Note is shipped with a default internal configuration that can be customized by merging a series of configuration files from various locations into the default values. This happens in the following order:

1. Unix and MacOS only: "`/etc/tpnote/tpnote.toml`"
2. The file the environment variable "`TPNOTE_CONFIG`" points to.
3. The user's configuration file:
   - Unix: "`~/.config/tpnote/tpnote.toml`"
   - Windows: "`C:\Users\<LOGIN>\AppData\Roaming\tpnote\config\tpnote.toml>`"
   - MacOS: "`/Users/<LOGIN>/Library/Application Support/tpnote`"
4. At startup all parent directories of the note file path "`<PATH>`'are searched for a marker file named" `.tpnote.toml`". If found, the document root moves from" `/`' to the found location. If present and its content is not empty, Tp-Note interprets the file's content as configuration file.
5. The file indicated by the command line parameter "`--config <FIlE>`".

When Tp-Note starts, it first merges all available configuration files into the default configuration. Then the resulting syntax is checked. If not correct, the last sourced configuration file is renamed (thus disabled) and Tp-Note starts with its internal default configuration. For debugging, you can print out the merge result with "`-V -b -d trace`".

To write a custom configuration file, first start with a complete default configuration you can generate by invoking Tp-Note with "`-C`":

```
tpnote -C ~/.config/tpnote/tpnote.toml
```

After modifying the concerned variables, delete step by step all the remaining variables to keep your configuration file as small as possible. Also make sure you delete the "`version`"variable at the beginning. As some Tp-Note upgrade include a configuration file structure change, a small configuration file increases the chance that it still merges.

Some filename and template related variables are grouped into a "`scheme`". The shipped configuration file lists two schemes: "`default`" and "`zettel`". The scheme used when creating a new note, is selected by the commend line option "`--scheme`", the environment variable "`TPNOTE_SCHEME`" or the configuration variable "`arg_default.scheme`". The scheme selected when synchronizing a Tp-Note header with its filename depends on the value of the header variable "`scheme:`" which defaults to "`default`" (cf. "`scheme_sync_default`").

Note, that the merging algorithm merges all values, except arrays. These are usually replaced by the subsequent configuration file. There is one exception though: top level arrays are also merged. An example to this is the top level array "`[[scheme]]`". In the following example we merge the variable "`extension_default = "txt"`"into the scheme "`default`" whereas all other variables remain untouched.

```
[[scheme]]
name='default'
[scheme.filename]
extension_default = "txt"
```

To add a custom scheme you must provide all variables:

```
[[scheme]]
name='my-custom-scheme'
[scheme.filename]
# Insert all variables here.
[scheme.tmpl]
# Insert all variables here.
```

The following example illustrates how non-top-level arrays are overwritten by the subsequent configuration file. The default configuration lists about 20 MIME types. After merging the following example, the configuration lists only the two MIME types "`jpeg`"and "`jpg`"in "`served_mime_types`".

```
[viewer]
```

```
served_mime_types = [
    ["jpeg", "image/jpeg"],
    ["jpg", "image/jpeg"],
]
```

8.1. Register your own text editor

There are two ways to modify the default file editor, Tp-Note launches when it starts: either you can modify the configuration file variables "`app_args.*.editor`" and "`app_args.*.editor_console`", or alternatively, you can set the "`TPNOTE_EDITOR`" environment variable (cf. examples in the chapter *ENVIRONMENT_VARIABLES* below).

The configuration file variables "`app_args.unix.editor`" and "`app_args.unix.editor_console`" define lists of external text editors to be launched for editing. The lists contain by default well-known text editor names and their command line arguments for Unix like operating systems. For other systems consult: "`app_args.windows.editor`", "`app_args.windows.editor_console`", "`app_args.macos.editor`" and "`app_args.macos.editor_console`". Tp-Note tries to launch every text editor in "`app_args.*.editor`"from the beginning of the list until it finds an installed text editor. When Tp-Note is started on a Linux console, the list "`app_args.*.editor_console`" is used instead. Here you can register text editors that do not require a graphical environment, e.g. "`vim`"or "`nano`". In order to use your own text editor, just place it at the top of the list. To debug your changes invoke Tp-Note with "`tpnote --debug debug --popup --edit`".

The following example showcases the configuration for the *Kate* file editor. The entry "`kate`" launches the binary, while the command line parameter "`--block`" guarantees, that the launched process blocks until the user closes the editor. Tp-Note detects the end of the process, checks if the title of the note files has changed in its YAML header and renames the note file if necessary.

```
[app_args]
unix.editor = [
  [
    'kate',
    '--block'
  ]
]
```

The equivalent configuration with environment variable:

```
TPNOTE_EDITOR="kate --block" tpnote
```

All items in the above list are subject to limited template expansion allowing to insert the value of environment variables. Consider the following example:

```
[app_args]
windows.editor = [
    [
    '{{ get_env(name="LOCALAPPDATA") }}\Programs\Microsoft VS Code
\Code.exe',
    "--new-window", "--wait",
    ]
]
```

When the configuration file is loaded, the above expression "`{{ get_env(name="LOCALAPPDATA") }}`"expands under Windows for a user with the username "`Joe`" to "`C:\User\Joe\AppData\Local`" resulting in:

```
[app_args]
windows.editor = [
    [
    'C:\User\Joe\AppData\Local\Programs\Microsoft VS Code\Code.exe',
    "--new-window", "--wait",
    ]
]
```

In general, when you configure Tp-Note to work with your text editor, make sure, that your text editor does not fork! You can check this by launching the text editor from the command line: if the command prompt returns immediately, then the file editor forks the process. On the other hand everything is OK, when the command prompt only reappears at the moment the text editor is closed. Many text editors provide an option to restrain from forking: for example the *VScode* file editor can be launched with the "`--wait`"option, *Vim* with "`--nofork`"or *Kate* with "`--block`".

However, Tp-Note also works with forking text editors. Although this should be avoided, there is a possible workaround. Observe the following example:

```
$ TPNOTE_EDITOR="kate" tpnote
/home/getreu/20230714-getreu--Note.md
$
```

In the above example Tp-Note launches the “`kate`”editor in a forking manner as the command line flag “`--block`”is missing. Internally the editor process launching returns immediately, leaving Tp-Note without any means to detect when exactly the user closes the editor. Hence, Tp-Note is not able to check if the user has changed the note’s header and no filename synchronization can occur afterwards.

As a workaround, you can manually trigger the filename synchronization after editing with “`tpnote --batch "$FILE"`”:

```
FILE=$(tpnote --batch) # Create the new note.
tpnote --view "$FILE"& # Launch Tp-Note's viewer.
kate "$FILE"           # Note, the prompt returns immediatly as the editor
 forks.
                       # After closing the editor when editing is done...
tpnote --batch "$FILE" # Synchronize the note's filename again.
```

Whereby “`FILE=$(tpnote --batch)`”creates the note file, “`kate "$FILE"`”opens the text editor and “`tpnote --batch "$FILE"`” synchronizes the filename after editing.

NB: Try to avoid forking at all cost. As mentioned above, most text editors have a command line flag to prevent the process from forking:

```
TPNOTE_EDITOR="kate --block" tpnote
```

Register a Flatpak Markdown editor

Flathub for Linux[4] is a cross-platform application repository that works well with Tp-Note. To showcase an example, we will add a Tp-Note launcher for the *Mark Text* Markdown text editor available as Flatpak package[5]. Before installing, make sure that you have set up Flatpack[6] correctly. Then install the application with:

```
sudo flatpak install flathub com.github.marktext.marktext
```

[4] https://www.flathub.org/home
[5] https://www.flathub.org/apps/details/com.github.marktext.marktext
[6] https://flatpak.org/setup/

To test, run *Mark Text* from the command line:

```
flatpak run com.github.marktext.marktext
```

Then place a Tp-Note configuration in its search path (e.g. "`~/.config/tpnote/tpnote.toml`") with the following content:

```
[app_args]
unix.editor = [ [ 'flatpak', 'run', 'com.github.marktext.marktext', ] ]
```

The structure of this variable is a list of lists. Every item in the outer list corresponds to one entire command line launching a different text editor, here *Marktext*. When launching, Tp-Note searches through this list until it finds an installed text editor on the system.

Save the modified configuration file. Next time you launch Tp-Note, the *Mark Text*-editor will open.

Register a console text editor running in a terminal emulator

In this setup Tp-Note launches the terminal emulator which is configured to launch the text editor as child process. Neither process should fork when they start (see above).

Here, some examples you can adjust to your needs and taste:

- *Neovim* in *Xfce4-Terminal*:

  ```
  [app_args]
  unix.editor = [
    [
      'xfce4-terminal',
      '--disable-server',
      '-x',
      'nvim',
      '+colorscheme pablo',
      '+set syntax=markdown',
    ],
  ]
  ```

- *Helix-editor* in *XFCE4-Terminal*:

```
[app_args]
unix.editor = [
  [
    'xfce4-terminal',
    '--disable-server',
    '-x',
    'hx',
  ],
]
```

- *Helix* in *LXTerminal*:

```
[app_args]
unix.editor = [
  [
    'lxterminal',
    '--no-remote',
    '-e',
    'hx',
  ],
]
```

- *Helix* in *Xterm*:

```
[app_args]
unix.editor = [
  [
    'xterm',
    '-fa',
    'DejaVu Sans Mono',
    '-fs',
    '12',
    '-e',
    'hx',
  ],
]
```

- *Helix* in *Alacritty*:

```
[app_args]
unix.editor = [
  [
    'alacritty',
    '-e',
```

```
      'hx',
    ],
  ]
```

8.2. Change the file extension for new note files

Tp-Note identifies the note's markup language by its file extension and renders the content accordingly (see "`filename.extensions`"variable). For example: the variable "`filename.extensions`" lists some extensions, that are regarded as Markdown files:

```
[[scheme]]
name = "default"
[scheme.filename]
extensions = [
  [ "txt", "Markdown" ],
  [ "md", "Markdown" ],
  [ "markdown", "Markdown" ],
  [ "markdn", "Markdown" ],
  [ "mdown", "Markdown" ],
  [ "mdtxt", "Markdown" ],
]
```

The default file extension for new note files is defined as:

```
[[scheme]]
name = "default"
[scheme.filename]
extension_default = 'md'
```

If you prefer rather the file extension "`.markdown`"for new notes, write a configuration file with:

```
[[scheme]]
name = "default"
[scheme.filename]
extension_default = 'markdown'
```

This modification does not change how the note file's content is interpreted - in this case as Markdown - because both file extensions "`.md`" and "`.markdown`" are rendered as "`Markdown`" according to "`filename.extensions`".

8.3. Configure the natural language detection algorithm

When creating a new header for a new or an existing note file, a linguistic language detection algorithm tries to determine in what natural language the note file is authored. Depending on the context, the algorithm processes as input: the header field “`title:`”or the first sentence of the text body. The natural language detection algorithm is implemented as a template filter named “`get_lang`”, which is used in various Tera content templates “`tmpl.*_content`” in Tp-Note’s configuration file. The filter “`get_lang`”is parametrized by the configuration variable “`tmpl.filter.get_lang`”containing a list of ISO 639-1 encoded languages, the algorithm considers as potential detection candidates, e.g.:

```
[[scheme]]
name = "default"
[scheme.tmpl]
filter.get_lang = [ 'en', 'fr', 'de', 'et' ]
```

As natural language detection is CPU intensive, it is advised to limit the number of detection candidates to 5 or 6, depending on how fast your computer is. The more language candidates you include, the longer the note file creation takes time. As a rule of thumb, with all languages enabled the creation of new notes can take up to 4 seconds on my computer. Nevertheless, it is possible to enable all available detection candidates with the pseudo language code “`+all`” which stands for “add all languages”:

```
[[scheme]]
name = "default"
[scheme.tmpl]
filter.get_lang = [ '+all', ]
```

Once the language is detected with the filter “`get_lang`”, it passes another filter called “`map_lang`”. This filter maps the result of “`get_lang`”- encoded as ISO 639-1 code - to an IETF language tag. For example, “`en`”is replaced with “`en-US`”or “`de`”with “`de-DE`”. This additional filtering is useful, because the detection algorithm can not figure out the region code (e.g. `-US` or `-DE`) by itself. Instead, the region code is appended in a separate processing step. Spell checker or grammar checker like [LTeX] rely on this region information, to work properly.

The corresponding configuration looks like this:

```
[[scheme]]
```

```
name = "default"
[scheme.tmpl]
filter.map_lang = [
    [ 'en', 'en-US', ],
    [ 'de', 'de-DE', ],
]
```

When the user's region setting - as reported from the operating system's locale setting - does not exist in above list, it is automatically appended as additional internal mapping. When the filter `map_lang` encounters a language code for which no mapping is configured, the input language code is forwarded as it is without modification, e.g. the input `fr` results in the output `fr`. Subsequent entries that differ only in the region subtag, e.g. "`['en', 'en- GB'], ['en', 'en-US']`" are ignored.

Note, that the environment variable "`TPNOTE_LANG_DETECTION`"- if set - takes precedence over the "`tmpl.filter.get_lang`"and "`tmpl.filter.map_lang`"settings. This allows configuring the language detection feature system-wide without touching Tp-Note's configuration file. The following example achieves the equivalent result to the configuration hereinabove:

```
TPNOTE_LANG_DETECTION="en-US, fr, de-DE, et" tpnote
```

If you want to enable all language detection candidates, add the pseudo tag "`+all`"somewhere to the list:

```
TPNOTE_LANG_DETECTION="en-US, de-DE, +all" tpnote
```

In the above example the IETF language tags "`en-US`"and "`de-DE`"are retained in order to configure the region codes "`US`" and "`DE`" used by the "`map_lang`" template filter.

For debugging observe the value of "`SETTINGS`" in the debug log with:

```
tpnote -d trace -b
```

If wished for, you can disable Tp-Note's language detection feature, by deleting all entries in the "`tmpl.filter.get_lang`" variable:

```
[[scheme]]
```

```
name = "default"
[scheme.tmpl]
filter.get_lang = []
```

Like above, you can achieve the same with:

```
TPNOTE_LANG_DETECTION="" tpnote
```

8.4. Localize the note's front matter

Be default, the front matter variable names are printed in English when creating new note files from templates. For example the header variable "`fm_subtitle`"is displayed as "`title:`" in the note's header.

This translation relation is defined in the configuration file variable "`scheme.tmpl.fm_vars.localization`". Consider the following simplified example:

```
[[scheme]]
name = "default"
fm_vars.localization = [
    ["fm_title", "title"],
    ["fm_subtitle", "subtitle"],
    ["fm_author", "author"],
    ["fm_date", "date"],
    ["fm_lang", "lang"],
]
```

To change the natural language of the displayed header variable names, modify the second column of the above table. Example:

```
[[scheme]]
name = "default"
fm_vars.localization = [
    ["fm_title", "Titel"],
    ["fm_subtitle", "Untertitel"],
    ["fm_author", "Autor"],
    ["fm_date", "Datum"],
    ["fm_lang", "Sprache"],
]
```

Keep in mind, that the templates do not change! Templates refer to a header variable with an identifier starting with “`fm_`”, The identifier corresponds to the first column of the above table.

8.5. Change the default markup language

Tp-Note’s core functionality, the management of note file headers and filenames, is markup language agnostic. However, there is one content template “`tmpl.annotate_file_content`” that generates a hyperlink. The hyperlink syntax varies depending on the markup language. Hence, you should not forget to modify the “`tmpl.annotate_file_content`” content template, when you change the default markup language defined in “`filename.extension_default`”.

Change default markup language to ReStructuredText

Tp-Note’s core function is a template system and as such it depends very little on the used markup language. The default templates are designed in a way that they contain almost no markup specific code. There is one little exception though. The following configuration variables affect the way new notes are created:

1. The file extension for new notes is defined as:

   ```
   [[scheme]]
   name = "default"
   [scheme.filename]
   extension_default='md'
   ```

   Overwrite this setting with the configuration file:

   ```
   [[scheme]]
   name = "default"
   [scheme.filename]
   extension_default='rst'
   ```

   Alternatively, the above can be achieved by setting the environment variable “`TPNOTE_EXTENSION_DEFAULT`”:

   ```
   TPNOTE_EXTENSION_DEFAULT="rst" tpnote
   ```

2. Replace the following line in the template “`tmpl.annotate_file_content`”that defines a hyperlink in Markdown format:

```
[{{ path | file_name }}](<{{ path | file_name }}>)
```

with the following line encoded in ReStructuredText syntax:

```
`<{{ path | file_name }}>`_
```

As a result, all future notes are created as "`*.rst`" files.

Change the markup language for one specific note only

You can change the Markup language of a specific note by adding the variable "`file_ext:`" to its YAML header. For example, for ReStructuredText add:

```
---
title:    some note
file_ext: rst
---
```

When Tp-Note triggers the next filename synchronization, the filename extension of the note file will change to "`.rst`". The above modification applies to the current note only.

8.6. Change the sort tag character set

Sort-tags for new notes are generated with the "`tmpl.*_filename`" templates. Before changing the sort-tag generation scheme in these templates, make sure to enable the right set of potential sort-tag characters.

In the default scheme, the digits "`0`"-"`9`", all lower case letters and the characters "`_`", "`-`", "`.`"are recognized as being part of a *sort tag* when they appear at the beginning of a filename. This set of characters can be modified with the "`filename.sort_tag.extra_chars`" configuration variable. If defined, the "`filename.sort_tag.separator`"(by default "`-`") marks the end of a sort tag without being part of it. In addition, one special character "`filename.sort_tag.extra_separator`"(by default "`'`") might be inserted by the filename template directly after the "`-`" to avoid ambiguity.

8.7. Customize the filename synchronization scheme

The filename synchronization scheme is fully customizable through *Tp-Note's filename templates*. To design such a custom scheme, start to set up your synchronization rules in the

“`tmpl.sync_filename`”template. Then adjust all “`tmpl.*_filename`”templates to comply with these rules. In order to verify your design, check that the following holds for any sequential application of one “`tmpl.*_filename`”template followed directly by the “`tmpl.sync_filename`”template: The latter should never change the filename initially set up by any “`tmpl.*_filename`” template.

Secondly, make sure that in filename templates “`tmpl.*_filename`”, sort-tags “`{{ path | file_sort_tag }}`” are never inserted directly. Instead, prepend the *sort_tag* with “`prepend(with_sort_tag=path|file_sort_tag)`” to the following expression, e.g.:

```
{{ fm_title | sanit | prepend(with_sort_tag=path|file_sort_tag) }}
```

The filter “`prepend(with_sort_tag=<...>)`” decides whether to insert the “`sort_tag.separator="-"`” and/or the “`sort_tag.extra_separator="'"`” characters. These heuristics enable Tp-Note to identify unequivocally sort-tags in filenames, which avoids potential cyclic filename change. Or, in other words: the “`tmpl.sync_filname`” template must always give the same result, even after repeated application.

To debug your “`tmpl.sync_filename`”template, create a test note file “`test.md`”and invoke Tp-Note with “`--debug trace`” and “`--batch`”:

```
tpnote --batch --debug trace test.md
```

8.8. Store new note files by default in a subdirectory

When you are annotating an existing file on disk, the new note file is placed in the same directory by default. To configure Tp-Note to store the new note file in a subdirectory, let’s say “`Notes/`”, instead, you need to modify the templates “`tmpl.annotate_file_filename`” and “`tmpl.annotate_file_content`”:

Replace in “`tmpl.annotate_file_filename`” the string:

```
{{ path | file_sort_tag }}
```

with:

```
Notes/{{ path | file_sort_tag }}
```

and in "`tmpl.annotate_file_content`":

```
[{{ path | filename }}](<{{ path | filename }}>)
```

with:

```
[{{ path | filename }}](<../{{ path | filename }}>)
```

8.9. Customize the built-in note viewer

Delay the launch of the web browser

By default, Tp-Note launches two external programs: some text editor and a web browser. If wished for, the configuration variable "`viewer.startup_delay`"allows delaying the launch of the web browser some milliseconds. This way the web browser window will always appear on top of the editor window. A negative value delays the start of the text editor instead.

Change the way how note files are rendered for viewing

Besides its core function, Tp-Note comes with several built-in markup renderer and viewer, allowing to work with different markup languages at the same time. The configuration file variable "`filename.extension`" determine which markup renderer is used for which note file extension. Depending on the markup language, this feature is more or less advanced and complete: *Markdown* (cf. "`Markdown`") is best supported and feature complete: It complies with the *Common Mark* specification. The *ReStructuredText* renderer (cf. "`Restructuredtext`") is quite new and still in experimental state. For all other supported markup languages Tp-Note provides a built-in markup source text viewer (cf. "`PlainText`") that shows the note as typed (without markup), but renders all hyperlinks to make them clickable. In case none of the above rendition engines suit you, it is possible to disable the viewer feature selectively for some particular note file extensions: just associate these extensions with the "`PlainTextNoViewer`" renderer. If you wish to disable the viewer feature overall, set the variable "`arg_default.edit = true`".

Change the HTML rendition template

After the markup rendition process, Tp-Note's built-in viewer generates its final HTML rendition through the customizable HTML templates "`tmpl_html.viewer`", "`tmpl_html.viewer_error`" and "`tmpl_html.exporter`". Unlike content

templates and filename templates, all HTML templates escape HTML critical characters in variables by default. To disable escaping for a specific variable, add the "`safe`" filter in last position of the filter chain. Please note, that in general, the "`safe`" filter is only recommended directly after the "`to_html`"and the "`markup_to_html`"filters, because these handle critical input by themselves. The following code example, inspired by the "`tmpl_html.viewer`" template, illustrates the available variables:

```
[tmpl_html]
viewer = '''
{%- set ext = fm_file_ext | default(value=extension_default ) -%}
<!DOCTYPE html>
<html lang="{{ fm_lang | default(value='en') }}">
<head>
<meta charset="utf-8">
<title>{{ fm_title }}</title>
<link rel="stylesheet" href="{{ viewer_doc_css_path }}">
<link rel="stylesheet" href="{{ viewer_highlighting_css_path }}">
  </head>
  <body>
  <pre class="doc-header">{{ doc_fm_text }}</pre>
  <hr>
  <div class="doc-body">
    {{ doc_body_text | markup_to_html(extension=ext) | safe }}
  </div>
  <script>{{ viewer_doc_js | safe }}</script>
</body>
</html>
'''
```

Specifically:

- "`{{ fm_* }}`"are the deserialized header variables. Note, that the header variables may be localized, e.g. "`Untertitel`". Nevertheless, in templates always use the English version, e.g. "`fm_subtitle`". All content template variables and filters are available. See section *Template variables* above.
- "`{{ viewer_doc_css_path }}`"is the CSS stylesheet path required to format a HTML rendition of a Tp-Note document. This path is hard-wired and it is understood by Tp-Note's internal web server.
- "`{{ viewer_highlighting_css_path }}`"is the CSS stylesheet path required to highlight embedded source code. This path is hard-wired and it is understood by Tp-Note's internal web server.

- "`{{ doc_fm_text }}`"is the raw UTF-8 copy of the header. Not to be confounded with the dictionary variable "`{{ fm_all }}`".
- "`{{ doc_body_text | markup_to_html(extension=ext) | safe }}`"is the note's body as HTML rendition. The parameter "`extension`"designates the markup language as specified in the "`filename.extensions-*`" variables .
- "`{{ doc_text | markup_to_html | safe }}`"is the note's raw text as HTML rendition with clickable hyperlinks.
- "`{{ viewer_doc_js | safe }}`" is the JavaScript browser code for live updates.
- "`{{ extension_default }}`" (c.f. section *Template variables*).
- "`{{ username }}`" (c.f. section *Template variables*).
- "`{{ lang }}`" (c.f. section *Template variables*).
- "`{{ my_val | to_html | safe }}`"is the HTML rendition of the "`my_val`" variable (c.f. section *Template filter*).

Alternatively, the header enclosed by "`<pre>...</pre>`" can also be rendered as a table:

```
<table class="fm">
  <tr>
  <th class="fmkey">title:</th>
  <th class="fmval"><b>
      {{ fm_title| default(value='') | to_html | safe }}</b>
  </th>
</tr>
  <tr>
  <th class="fmkey">subtitle:</th>
  <th class="fmval">
      {{ fm_subtitle | default(value='') | to_html | safe }}
  </th>
</tr>
{% for k, v in fm_all| remove(key='fm_title')|
                       remove(key='fm_subtitle')|
%}
  <tr>
  <th class="fmkeygrey">{{ k }}:</th>
  <th class="fmvalgrey">{{ v | to_html | safe }}</th>
</tr>
{% endfor %}
</table>
```

The error page template “`tmpl_html.viewer_error`” (see below) does not provide “`fm_*`” variables, because of possible header syntax errors. Instead, the variable “`{{ doc_error }}`” contains the error message as raw UTF-8 and the variable “`{{ doc_text | markup_to_html | safe }}`”the HTML rendition of the text source with clickable hyperlinks:

```
[tmpl_html]
viewer_error = '''
<!DOCTYPE html>
<html lang=\"en\">
<head>
<meta charset=\"UTF-8\">
<title>Syntax error</title>
</head>
<body>
<h3>Syntax error</h3>
<p> in note file: <pre>{{ path }}</pre><p>
<div class=\"note-error\">
<hr>
<pre>{{ doc_error }}</pre>
<hr>
</div>
{{ doc_text | markup_to_html | safe }}y
<script>{{ viewer_doc_js | safe }}</script>
</body>
</html>
'''
```

Customize the built-in HTML exporter

Customizing Tp-Note’s HTML export function works the same way as customizing the built-in viewer. There are some slight differences though: The role of the “`tmpl_html.viewer`” template - discussed above - is taken over by the “`tmpl_html.exporter`” template:

```
[tmpl_html]
exporter = '''
{%- set ext = fm_file_ext | default(value=extension_default ) -%}
<!DOCTYPE html>
<html lang="{{ fm_lang | default(value='en') }}">
<head>
<meta charset="utf-8">
<title>{{ fm_title }}</title>
<style>
```

```
{{ exporter_doc_css | safe }}
{{ exporter_highlighting_css | safe }}
</style>
</head>
<body>
  <pre class="doc-header">{{ doc_fm_text }}</pre>
  <hr>
  <div class="doc-body">
    {{ doc_body_text| markup_to_html(extension=ext) | safe }}
  </div>
</body>
</html>
'''
```

In this template the same *Tera* variables as in "`tmpl_html.viewer`" are available, with one exception "`{{ note_js }}`", which does not make sense in this context. As the exporter prints possible rendition error messages on the console, there is no equivalent to the "`tmpl_html.viewer_error`"template. Note, in contrast to the previous "`tmpl_html.viewer`"example, the source code highlighting CSS code is now embedded into the HTML output with:

```
<style>
{{ exporter_doc_css | safe }}
{{ exporter_highlighting_css | safe }}
</style>
```

Note, the "`safe`" filter disables the escaping of critical characters in the CSS input. We have no security concerns in this context, because we have full control over the CCS input coming from the configuration file variables "`tmpl_html.exporter_doc_css`" and "`tmpl_html.exporter_highlighting_theme`".

8.10. Choose your favourite web browser as note viewer

Once the note is rendered into HTML, Tp-Note's internal HTTP server connects to a random port at the "`localhost`" interface where the rendition is served to be viewed with a web browser. Tp-Note's configuration file contains a list "`app_args.unix.browser`"with common web browsers and their usual location on Unix like operating systems. For other systems consult "`app_args.windows.browser`"and "`app_args.macos.browser`". This list is executed top down until a web browser is found and launched. If you want to view your notes with a different web browser, simply modify the "`app_args.unix.browser`"list and put your favourite web browser on top. Alternatively,

you can set the "`TPNOTE_BROWSER`" environment variable (cf. examples in the chapter *ENVIRONMENT_VARIABLES* below).

In case none of the listed browsers can be found, Tp-Note switches into a fallback mode with limited functionality, where it tries to open the system's default web browser. A disadvantage is, that in fall back mode Tp-Note is not able to detect when the user closes the web browser. This might lead to situations, where Tp-Note's internal HTTP server shuts down to early. In order to check if Tp-Note finds the selected web browser as intended, invoke Tp-Note with "`tpnote --debug debug --popup --view`".

9. TEMPLATES

All *TP-Note*'s workflows are customizable through its templates which are grouped in the "`[scheme.tmpl]`" and in the "`[scheme.tmpl_html]`" section of Tp-Nots's configuration file. This chapter deals with "`[scheme.tmpl]`" templates which are responsible for generating Tp-Note files. "`[scheme.tmpl_html]`"templates concern only Tp-Note's viewer feature and are discussed in the chapters: Customize the built-in note viewer_ and *Choose your favourite web browser as note viewer*.

Tp-Note captures and stores its environment in *Tera variables*. For example, the variable "`{{ dir_path }}`" is initialized with the note's target directory. The variable "`{{ clipboard }}`" contains the content of the clipboard. To learn more about Tera variables, launch Tp-Note with the "`--debug trace`"option and observe what information it captures from its environment.

9.1. Template types

The content of a new note is composed by one of Tp-Note's internal customizable templates, hence the name Tp-Note, where *Tp* stands for "template". Which of the internal templates is applied depends on the context in which Tp-Note is invoked: e.g. the template for clipboard text input is called "`tmpl.from_clipboard_content`". If the clipboard contains text with a YAML header, the template "`tmpl.from_clipboard_yaml_content`" is used.

In total, there are 5 different "`tmpl.*_content`" templates:

- "`tmpl.from_dir_content`"
- "`tmpl.from_clipboard_content`"
- "`tmpl.from_clipboard_yaml_content`"
- "`tmpl.from_text_file_content`"
- "`tmpl.annotate_file_content`"

In general, the templates are designed in a way, that the text input stream - usually originating from the clipboard - ends up in the body of the note file, whereas the environment - such as the username - ends up in the header of the note file.

Once the content of the new note is set by one of the content templates, another template type comes into play: the so-called *filename template*. Each content template has a corresponding filename template, e.g.:

- "`tmpl.from_dir_filename`"
- "`tmpl.from_clipboard_filename`"
- "`tmpl.from_clipboard_yaml_filename`"
- "`tmpl.from_text_file_filename`"
- "`tmpl.annotate_file_filename`"
- "`tmpl.sync_filename`" (no corresponding content template)

As the name suggests, the role of a filename template is to determine the filename of the new note. This is done by evaluating (deserializing) it's YAML header. The values of the note's YAML header fields are can be accessed in filename templates through various "`{{ fm_<key> }}`" dynamically created template variables. For example the value of the YAML header field "`title:`" can be accessed with "`{{ fm_title }}`". Once the filename is set, Tp-Note writes out the new note on disk.

Most of the above templates are dedicated to the creation of new note files. However, two of them have a special role: *prepend header to text file* and *synchronize filename*:

- *Prepend header to text file* (new feature in Tp-Note v1.16.0): When Tp-Note opens a regular text file without a YAML header, a new header is prepended automatically. It's data originates mainly form the filename of the text file. The templates applied in this use case are: "`tmpl.from_text_file_content`" and "`tmpl.from_text_file_filename`".
- *Synchronize filename*: This function mode is invoked when [Tp-Note] opens an existing note file, after it's YAML header is evaluated. The extracted header information is then applied to the "`tmpl.sync_filename`" template and the resulting filename is compared with the actual filename on disk. If they differ, [Tp-Note] renames the note file. Note, the "`tmpl.sync_filename`" template operates on its own without a corresponding content template.

Note, that in the operation mode *synchronize filename*, the header data overwrites the filename of the note, whereas in the operation mode *prepend header* the

filename data is copied into the new prepended header. Keep in mind, that even in the latter mode the filename might change slightly. This is because after the header creation with the "`tmpl.from_text_file_content`" template, the "`tmpl.from_text_file_filename`" template is applied, which might cause a slight filename modification due to its sanitization filters (cf. "`sanit()`" in the section *Template filters*).

You can disable the *prepend header* feature by setting the configuration file variable "`arg_default.add_header = false`". To disable all filename synchronization, set "`arg_default.no_filename_sync = true`". This guarantees, that Tp-Note will never change neither the filename nor the YAML header of an existing file.

For a more detailed description of templates and their defaults, please consult the "`const`" definitions in Tp-Note's source code files "`config.rs`" and "`note.rs`" in the directory "`tpnote-lib/src/`".

9.2. Template variables

All Tera template variables and functions[7] can be used within Tp-Note's templates. For example "`{{ get_env(name='LANG') }}'` gives you access to the" `LANG`' environment variable.

In addition, Tp-Note defines the following variables:

- "`{{ path }}`" is the canonicalized fully qualified path name corresponding to Tp-Note's positional command line parameter "`<path>`". If none was given on the command line, "`{{ path }}`" contains the current working directory path.
- "`{{ dir_path }}`" is identical to "`{{ path }}`" with one exception: if "`{{ path }}`" points to a file, the last component (the file name) is omitted and only the directory path is retained. If "`{{ path }}`" points to a directory, "`{{ dir_path }}`" equals "`{{ path }}`".
- "`{{ note_fm_text }}`": is the header as raw text of the file "`{{ path }}`" points to. Note, this variable is only available in the templates "`from_text_file_*`", "`sync_filename`" and the HTML templates below.
- "`{{ note_body_text }}`": is the content of the file "`{{ path }}`" points to. If the file does not start with a front matter, this variable holds the whole content. Note, this variable is only available in the templates "`from_text_file_*`", "`sync_filename`" and the HTML templates below.

[7] https://tera.netlify.com/%20docs/#templates

- "`{{ note_file_date }}`": is the file system creation date of the file "`{{ path }}`" points to. Note, this variable is only available in the templates "`from_text_file_*`", "`sync_filename`" and the HTML templates below.
- "`{{ clipboard }}`"is the complete clipboard text. In case the clipboard's content starts with a YAML header, the latter does not appear in this variable.
- "`{{ clipboard_header }}`"is the YAML section of the clipboard data, if one exists. Otherwise: empty string.
- "`{{ stdin }}`"is the complete text content originating from the input stream "`stdin`". This stream can replace the clipboard when it is not available. In case the input stream's content starts with a YAML header, the latter does not appear in this variable.
- "`{{ stdin_header }}`" is the YAML section of the input stream, if one exists. Otherwise: empty string.
- "`{{ extension_default }}`"is the default extension for new notes (can be changed in the configuration file),
- "`{{ username }}`" is the content of the first non-empty environment variable: "`TPNOTE_USER`", "`LOGNAME`", "`USER`" or "`USERNAME`".
- "`{{ lang }}`"contains the user's language tag as defined in RFC 5646[8]. Not to be confused with the UNIX "`LANG`"environment variable from which this value is derived under Linux/macOS. Under Windows, the user's language tag is queried through the WinAPI. If defined, the environment variable "`TPNOTE_LANG`"overwrites the value of "`{{ lang }}`" (all operating systems).

The following "`{{ fm_* }}`" variables are typically generated, *after* a content template was filled in with data: For example a field named "`title:`" in the content template "`tmpl.from_dir_content`" will generate the variable "`fm_title`" which can then be used in the corresponding "`tmpl.from_dir_filename`"filename template. "`{{ fm_* }}`"variables are generated dynamically. This means, a YAML front-matter variable "`foo:`"in a note will generate a "`{{ fm_foo }}`"template variable. On the other hand, a missing "`foo:`"will cause "`{{ fm_foo }}`"to be undefined. Please note, that the header variables may be localized, e.g. "`Untertitel:`". Nevertheless, in templates always use the English version, e.g. "`fm_subtitle`".

It is to be observed that "`{{ fm_* }}`"variables are only available in filename templates and in the "`tmpl.from_clipboard_yaml_content`" content template.

- "`{{ fm_title }}`" is the "`title:`" as indicated in the YAML front-matter of the note.

[8] http://www.rfc-editor.org/rfc/rfc5646.txt

- "`{{ fm_subtitle }}`"is the "`subtitle:`"as indicated in the YAML front matter of the note.
- "`{{ fm_author }}`"is the "`author:`"as indicated in the YAML front-matter of the note.
- "`{{ fm_lang }}`" is the "`lang:`" as indicated in the YAML front-matter of the note.
- "`{{ fm_file_ext }}`" holds the value of the optional YAML header variable "`file_ext:`" (e.g. "`file_ext: rst`").
- "`{{ fm_sort_tag }}`": The sort tag variable as defined in the YAML front matter of this note (e.g. "`sort_tag: '20200312'`").
- "`{{ fm_all }}`": is a collection (map) of all defined "`{{ fm_* }}`"variables. It is used in the "`tmpl.from_clipboard_yaml_content`" template, typically in a loop like:

```
{% for key, value in fm_all %}{{ key }}: {{ value | json_encode }}
{% endfor %}
```

Important: there is no guarantee, that any of the above "`{{ fm_* }}`"variables are defined! Depending on the last content template result, certain variables might be undefined. Please take into consideration, that a defined variable might contain the empty string "`""`". Creating a new note file with a content template, the note's header is parsed into "`{{ fm_* }}`" variables. The latter are then type checked according configurable rules. The rules are defined in "`tmpl.filter.assert_precondition`"

For a more detailed description of the available template variables, please consult the "`const`" definitions in Tp-Note's source code file "`note.rs`".

9.3. Template filters

In addition to *Tera*'s built-in filters[9], Tp-Note comes with some additional filters, i.e.: "`append(newline=true)`", "`append(with=...)`", "`cut`", "`file_copy_counter`", "`file_ext`", "`file_name`", "`file_sort_tag`", "`file_stem`", "`get_lang`", "`heading`", "`insert(key=..., value=...)`", "`link_dest`", "`link_text`", "`link_title`", "`map_lang`", "`prepend`", "`prepend(newline=true)`", "`prepend(with=...)`", "`prepend(with_sort_tag=...)`", "`remove(key=)`" "`sanit`", "`to_html`", "`to_yaml`", "`to_yaml(key=...)`", "`to_yaml(tab=...)`" and "`trim_file_sort_tag`".

[9] https://tera.netlify.app/docs/#built-in-filters

A filter is always used together with a variable. Here are some examples:

- "`{{ path | file_name }}`"returns the final component of "`{{ path }}`". If "`{{ path }}`"points to a file, the filter returns the complete filename including its sort tag, stem, copy-counter, dot and extension. If the "`<path>`"points to a directory, the filter returns the final directory name.
- "`{{ path | file_sort_tag }}`"is the sort tag (numerical filename prefix) of the final component of "`{{ path }}`", e.g. "`01-23_9`"or "`20191022`". It is similar to "`{{ path | file_name }}`"but without returning its stem, copy-counter and extension.
- "`{{ path | file_sort_tag | assert_valid_sort_tag }}`" does not change the above output, but the filter asserts at runtime, that the resulting type is either "`String`" or "`Number`" and that all characters are part of the set "`filename.sort_tag.extra_chars`". The additional runtime check simplifies template debugging.
- "`{{ path | file_stem }}`"is similar to "`{{ path | file_name }}`"but without its sort tag, copy-counter and extension. Only the stem of "`{{ path }}`"s last component is returned.
- "`{{ path | file_copy_counter }}`" is similar to "`{{ path | file_name }}`" but without its sort tag, stem and extension. Only the copy counter of "`{{ path }}`"'s last component is returned.
- "`{{ path | file_ext }}`"is "`{{ path }}`"s file extension without dot (period), e.g. "`txt`" or "`md`".
- "`{{ path | file_ext | prepend(with='.') }}`"is "`{{ path }}`"s file extension with dot (period), e.g. "`.md`" or "`.md`".
- "`{{ path | trim_file_sort_tag }}`"returns the final component of "`path`" which might be a directory name or a file name. Unlike the "`file_name`"filter (which also returns the final component), "`trim_file_sort_tag`" trims the sort tag if there is one.
- "`{{ dir_path | find_last_created_file | incr_sort_tag(default="") }}`" searches "`dir_path`"for the most recently created Tp-Note file, extracts the sort-tag from this file, increments the sort-tag and returns the result. If the incrementation fails, the "`default`" value is returned. This can happen, when the input sort-tag contains characters of the set "`tmpl.filter.incr_sort_tag.default_if_contains`". Or, if the to be incremented counter (a sequential sort-tag usually has more than one counter) has more than "`tmpl.filter.incr_sort_tag.default_if_greater`" digits.

- "`{{ dir_path | trim_file_sort_tag }}`" returns the final component of "`dir_path`" (which is the final directory name in "`{{ path }}`"). Unlike the "`file_name`" filter (which also returns the final component), "`trim_file_sort_tag`" trims the sort tag if there is one.
- "`{{ clipboard | cut }}`" is the first 200 bytes from the clipboard.
- "`{{ clipboard | heading }}`"is the clipboard's content until the end of the first sentence, or the first newline.
- "`{{ clipboard | link_text }}`" is the name of the first Markdown or ReStructuredText formatted link in the clipboard.
- "`{{ clipboard | link_dest }}`" is the URL of the first Markdown or ReStruncturedText formatted link in the clipboard.
- "`{{ clipboard | link_title }}`" is the title of the first Markdown or ReStruncturedText formatted link in the clipboard.
- "`{{ username | capitalize | to_yaml(key='author',tab=12) }}`" is the capitalized YAML encoded username. As all YAML front-matter is YAML encoded, the "`to_yaml`"filter must be appended to any template variable placed in the front-matter block. The "`key='author'`"parameter prepends the key to the capitalized username, e.g.: "`autor: John`". Note, the first letter of "`John`"starts at the tabulator position "`tab=12`".
- "`{{ fm_subtitle | sanit }}`"is the note's subtitle as defined in its front matter, sanitized in a file system friendly form. Special characters are omitted or replaced by "`-`" and "`_`". See the section *Filename template convention* for more details about this filter.
- "`{{ fm_title | sanit | prepend(with_sort_tag=path| file_sort_tag) }}`" is the note's title as defined in its front-matter. Same as above, but the title string is prepended with the note's *sort_tag* and with a "`filename.sort_tag.separator`" (by default "`-`"). Eventually, a second "`filename.sort_tag.extra_separator`"(by default "`''`") is inserted after the first to guarantee, that one of the separators unequivocally marks the end of the *sort_tag*. This might be necessary to avoid ambiguity in case the "`fm_title`"starts with a character defined in the "`filename.sort_tag.extra_chars`" set.
- "`{{ fm_all | remove(key='fm_title') | remove(key='fm_author') | to_yaml }}`" renders the collection (map) "`fm_all`", exclusive of the variables "`fm_title`"and "`fm_author`"to YAML. Note, that the filter "`to_yaml`"has no parameter "`key`" in this context.
- "`{{ fm_all | insert(key='fm_author', value='Getreu') | to_yaml}}`" takes the collection (map) "`fm_all`", inserts the key/value

“`fm_author`”/“`Jens`” and renders the result into YAML. Note, that the filter “`to_yaml`” has no parameter “`key`” in this context.

- “`{{ fm_all | to_yaml | append(newline=true) }}`”renders the collection (map) “`fm_all`”into YAML. If the collection is empty, the result is the empty string. Otherwise, the YAML rendition is appended with a newline character.
- “`{{ fm_all | to_html | safe }}`”renders the collection (map) “`fm_*`”into HTML. The “`to_html`”must be followed by a “`safe`”filter to pass through the HTML formatting of objects and arrays.
- “`{{ note_body_text | get_lang }}`”determines the natural language of the variable “`{{ note_body_text }}` and returns the result as ISO 639-1 language code. The template filter”`{{ get_lang }}`’ can be configured with the configuration file variable “`tmpl.filter.get_lang`”. The latter defines a list of ISO 639-1 codes, the detection algorithm considers as possible language candidates. Keep this list as small as possible, because language detection is computationally expensive. A long candidate list may slow down the note file creation workflow. If the detection algorithm can not determine the language of “`{{ note_body_text }}`”, the filter “`get_lang`”returns the empty string.
- “`{{ note_body_text | get_lang | map_lang }}`” maps the detected ISO 638-1 language code to a complete IETF BCP 47 language tag, usually containing the region subtag. For example the input “`en`” results in “`en-US`”. This additional mapping is useful because the detection algorithm can not determine the region automatically. The mapping can be configured by adjusting the configuration file variable “`tmpl.filter.map_lang`”. If a language is not listed in the “`tmpl.filter.map_lang`”filter configuration, the input is passed through, e.g. “`fr`” results in “`fr`”, or, the empty string results in an empty string.
- “`{{ note_body_text | get_lang | map_lang(default=lang) }}`”adds an extra mapping for the “`map_lang`”filter: when the input of the “`map_lang`”filter is the empty string, then it’s output becomes the value of the “`{{ lang }}`” variable.

9.4. Content template conventions

Tp-Note distinguishes two template types: content templates are used to create the note’s content (front-matter and body) and the corresponding filename templates “`tmpl.*_filename`” are used to calculate the note’s filename. By convention, content templates appear in the configuration file in variables named “`tmpl.*_content`”.

Strings in the front matter section of content templates are YAML encoded. Therefore, all variables used in the front-matter must pass an additional “`to_yaml()`”-filter. For example,

the variable "`{{ dir_path | file_stem() }}`" becomes "`{{ dir_path | file_stem() | to_yaml(key='title') }}`" or, shorter: "`{{ dir_path | file_stem | to_yaml(key='title') }}`".

When given with a key, the "`to_yaml(key='...')`" filter accepts any input type, whereas the short form "`to_yaml()`" requires an "`Value::Object`" type as input. The latter is often followed be the "`append(newline=true)`" filter appending a newline.

9.5. Filename template conventions

By convention, filename templates appear in the configuration file in variables named "`tmpl.*_filename`". When a content template creates a new note, the corresponding filename template is called afterwards to calculate the filename of the new note. Please note that, the filename template "`tmpl.sync_filename`" has a special role as it synchronizes the filename of existing note files. Besides this, as we are dealing with filenames we must guarantee, that the filename templates produce only file system friendly characters. For this purpose Tp-Note provides the additional Tera filter "`sanit`":

The "`sanit()`" filter transforms a string in a file system friendly from. This is done by replacing forbidden characters like "`?`" and "`\\`" with "`_`" or space. This filter can be used with any variable, but is most useful with filename templates. For example, in the "`tmpl.sync_filename`" template, we find the expression "`{{ subtitle | sanit }}`". Note that the filter recognizes strings that represent a so-called dot file name and treats them a little differently by prepending them with an apostrophe: a dot file is a file whose name starts with "`.`" and that does not contain whitespace. It may or may not end with a file extension. The apostrophe preserves the following dot from being filtered.

The "`prepend(with_sort_tag=<...>`" filter is similar to the "`prepend(with=<...>`" filter, with two exceptions:

1. If "`filename.sort_tag.separator`" is defined (by default "`-`"), it is automatically inserted between the sort-tag and the input string.
2. In some cases an additional separator "`filename.sort_tag.extra_separator`" (by default "`'`") may be inserted as well.

Both separators guarantee that the end of a sort-tag is detected unequivocally. For example, when the input string starts with a digit "`0123456789`" or "`-_`", the string is prepended with `-'`, e.g. "`1-The Show Begins`" becomes "`'1-The Show Begins`". The "`prepend(with_sort_tag=<...>)`" filter must be applied to the first variable,

e.g. “`{{ fm_title | sanit | prepend(with_separator=path| file_sort_tag )}`”. This way, it is always possible to univocally distinguish the sort-tag from the rest of the filename. Note, the default sort-tag separators can be changed with the configuration variables “`filename.sort_tag.separator`” and “`filename.sort_tag.extra_separator`”. For more details please consult the *Customize the filename synchronization scheme* chapter.

In filename templates most variables must pass the “`sanit`”filter. Exception to this rule are sort-tag expressions like “`{{ path | file_sort_tag }}`”and “`{{ dir_path | file_sort_tag }}`”. As the latter are guaranteed to contain only the file system friendly characters “`0123456789 -_`”, no additional filtering is required. Please note, that in this case a “`sanit`”-filter would needlessly restrict the value range of sort-tags because they may contain characters, which the “`sanit`”-filter screens out when they appear in leading or trailing position. For this reason one must not use the “`sanit`”-filter together with “`{{ path | file_sort_tag }}`” or “`{{ dir_path |file_sort_tag }}`”.

10. SECURITY AND PRIVACY CONSIDERATIONS

As discussed above, Tp-Note’s built-in viewer sets up an HTTP server on the “`localhost`” interface with a random port number.

For security reasons, Tp-Note limits the set of files the viewer is able to publish. To summarize, a file is only served:

1. when it is referenced in one of the currently viewed Tp-Note files,
2. when its file extension is registered with the “`viewer.served_mime_type`” list,
3. if the number of so far viewed Tp-Note files, “`viewer.displayed_tpnote_count_max`” is not exceeded,
4. when it’s located under a directory containing a marker file named “`.tpnote.toml`” (without marker file this condition is void).

The HTTP server runs as long as the launched web browser window is open. Note, that the server not only exposes the displayed note file, but also all referenced inline images and other linked TP-Note files. Internally, the viewer maintains a list of *referenced local URLs*. For security reasons, only listed files are served. To limit data exfiltration in case an attacker gains access to an account on your machine, the number of served Tp-Note files is limited by the configurable value “`viewer.displayed_tpnote_count_max`”.

In addition to the above quantitative restriction, Tp-Note’s built-in viewer serves only files whose file extensions are registered with the “`viewer.served_mime_type`”

configuration file variable. The latter allows disabling the *follow links to other Tp-Note files* feature by removing all "`text/*`" mime types from that list.

Another security feature is the "`.tpnote.toml`"marker file. When Tp-Note opens a note file, it checks all directories above, one by one, until it finds the marker file "`.tpnote.toml`". Tp-Note's viewer will never serve a file located outside the root directory and its children. When no "`.tpnote.toml`"file is found, the root directory is set to "`/`", which disables this security feature.

As Tp-Note's built-in viewer binds to the "`localhost`"interface, the exposed files are in principle accessible to all processes running on the computer. As long as only one user is logged into the computer at a given time, no privacy concern is raised: any potential attacker must be logged in, in order to access the `localhost` HTTP server.

This is why on systems where multiple users are logged in at the same time, it is recommended to disable Tp-Note's internal HTTP server by setting the configuration file variable "`arg_default.edit = true`". Alternatively, you can also compile Tp-Note without the "`viewer`" feature. Note, that even if the viewer feature disabled, the "`--export`" command line option still works: This allows the authorized user to render the note to HTML manually.

Summary: As long as Tp-Note's built-in note viewer is running, the note file and all its referenced (image) files are exposed to all users logged into the computer at that given time. This concerns only local users, Tp-Note never exposes any information to the network or on the Internet.

11. ENVIRONMENT VARIABLES

LANG

> Tp-Note stores the user's locale settings - originating from the environment variable "`LANG`" (or the Windows registry) - in the template variable "`{{ lang }}`". When the environment variable "`TPNOTE_LANG`"is set, it overwrites the locale setting stored in "`{{ lang }}`". "`man locale`" describes the data format of "`LANG`", a typical value is "`en_GB.UTF-8`".

TPNOTE_CONFIG

> When set, the environment variable replaces the default path where Tp-Note loads or stores its configuration file. It has the same effect as the command line option "`--config`". If both are present, that latter takes precedence.

TPNOTE_LANG

Tp-Note stores the user's locale settings - originating from the environment variable "`LANG`" (or the Windows registry) - in the template variable "`{{ lang }}`". When the environment variable "`TPNOTE_LANG`"is set, it overwrites the locale setting stored in "`{{ lang }}`". Unlike "`LANG`", the environment variable "`TPNOTE_LANG`"is encoded as IETF BCP 47 language tag, e.g. "`en-US`".

TPNOTE_LANG_DETECTION

If set, this variable overwrites the configuration file variables "`tmpl.filter.get_lang`" and "`tmpl.filter.map_lang`", thus selecting potential language candidates for Tp-Note's natural language detection. The string contains a comma and space separated list of ISO 63901 codes, e.g. "`fr`"or IETF BCP 47 tags, e.g. "`fr-FR`". Here is an example of a complete string: "`de-DE, en, fr-FR, hu`". The user's default locale "`{{ lang }}`"is automatically added to the list. Note, that the language detection algorithm determines only the language subtag, e.g. "`en`". The region subtag will be added as indicated in your configuration. Subsequent entries that differ only in the region subtag, e.g. "`en-GB, en-US`" are ignored.

The empty string disables the automatic language detection.

```
TPNOTE_LANG_DETECTION="" tpnote
```

For debugging observe the value of "`SETTINGS`" in the debug log:

```
TPNOTE_LANG_DETECTION="de-DE, en, fr-FR" tpnote -d trace -b
```

TPNOTE_BROWSER

If set, this variable take precedence over the configuration file variable "`app_args.browser`". While the latter is a list describing how to invoke various web browsers, "`TPNOTE_BROWSER`"contains a string invoking one particular browser, exactly as one would do in a shell: the whitespace separated tokens list contains: the path name of the application, and all its flags and options. For example:

```
TPNOTE_BROWSER="chromium --new-window --incognito" tpnote
```

The above instructs Tp-Note to start the web browser "`chromium`" with the flags "`--new-window`" and "`--incognito`". Unlike in a shell, the backslash and quote characters have no special meaning. Instead, all tokens are *percent encoded*, e.g. "`my path`" becomes "`my%20path`".

The empty string disables the launch of the browser the same way as "`--edit`":

```
TPNOTE_BROWSER="" tpnote
```

is equivalent to:

```
tpnote --edit
```

TPNOTE_EDITOR

If set, and you are working on a graphical desktop, this variable takes precedence over the configuration file variable "`app_args.editor`". While the latter is a list describing how to invoke various file editors, "`TPNOTE_EDITOR`" contains a string invoking one particular file editor, exactly as one would do on a shell: the whitespace separated tokens list contains: the path name of the application, and all its flags and options. For example:

```
TPNOTE_EDITOR="geany -sim" tpnote
```

The above instructs Tp-Note to start the editor "`geany`" with the flags "`-sim`". Unlike with shell tokens, the backslash and quote characters have no special meaning. Instead, all tokens are *percent encoded*. Consider the following example where the space character is expressed as "`%20`":

```
TPNOTE_EDITOR="geany -sim -c ~/my%20config/" tpnote
```

The empty string disables the launch of the editor the same way as the command line option "`--view`" does:

```
TPNOTE_EDITOR="" tpnote
```

is equivalent to:

```
tpnote --view
```

TPNOTE_EDITOR_CONSOLE

If set, and you are working on a virtual console, this variable takes precedence over the configuration file variable "`app_args.editor_console`", which defines the command line parameters for invoking a terminal based text editor, such as Emacs, Vim or Helix. Otherwise, the syntax and the operation are the same as with "`TPNOTE_EDITOR` hereinabove". Example of use:

```
sudo TPNOTE_EDITOR_CONSOLE="nvim" tpnote
```

TPNOTE_EXTENSION_DEFAULT

If set, this variable takes precedence over the configuration file variable "`filename.extension_default`", which defines the file extension of new note files. In order to activate the appropriate markup renderer make sure, that the value given here is listed in "`filename.extensions`".

TPNOTE_SCHEME

If set, this variable takes precedence over the configuration file variable "`arg_default.scheme`", which defines the scheme used when creating new note file.

TPNOTE_USER, LOGNAME, USER, USERNAME

The template variable "`{{ username }}`"is the content of the first non-empty environment variable: "`TPNOTE_USER`", "`LOGNAME`", "`USER`" or "`USERNAME`".

12. EXIT STATUS

The exit status is "`0`"when the note file was processed without error or "`1`"otherwise. If Tp-Note can not read or write its configuration file, the exit status is "`5`".

When "`tpnote -n -b <FILE>`"returns the code "`0`", the note file has a valid YAML header with a "`title:`"field. In addition, when "`tpnote -n -b -x - <FILE>`"returns the code "`0`", the note's body was rendered without error.

13. RESOURCES

Tp-Note it hosted on:

- Gitlab: https://gitlab.com/getreu/tp-note.
- Github (mirror): https://github.com/getreu/tp-note and on

14. COPYING

14.1. Contribution

15. AUTHORS

Jens Getreu `<getreu@web.de>`