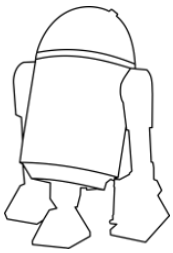
Paperwork Hacking Guide

Contents

1 This is not the document you're looking for

This is an entirely technical document. If you're not looking to contribute to Paperwork's code, this is not the document you're looking for.



This document assumes you're already familiar with Git, Python 3, and GitHub.

2 Please discuss before doing

You should always discuss what you want to do before doing it. Main reasons are:

- If someone else is already working on it, work will be done twice. And someone's work will have to be thrown away.
- You can get confirmation that what you want to do is in line with Paperwork's goals (easy to use, ergonomic, etc).
- You can get suggestions. It can strongly reduce the risk of your first pull request being rejected.

The easiest way is to open a ticket on GitHub[1]. Just indicate you intend to work on it, and the ticket will be assigned to you.

3 Git branches

In the repository, there are usually the following branches:

- *stable* : Matches the latest release version. May also include some bug fixes that will be included in the next update (x.y.z ; ex: 1.2.2). No new features allowed.
- *unstable* : Will become the next version (x.y.0 ; ex: 1.3.0) of Paperwork. New features go here. This branch hasn't been tested yet and things may be broken.
- *testing* : When the unstable branch contains all the wanted features, the branch *testing* is created from the branch *unstable*. Only bug fixes, translations updates and documentation updates are allowed in this branch. This branch is temporary and usually lives for about 1 month. After 1 month, the version is released, and this branch is merged in the branch *stable*.
- *wip-<XXX>* : Branches for work in progress. They are temporary. There are no specific rules for those branches.

4 Coding rules

4.1 PEP8

Stick to the PEP8 as much as possible. Pull requests that does not respect the PEP8 will be rejected.

- Lines are at most 80 characters long (not kidding.)
- Indentation is done using 4 spaces

If you see code that does not respect the PEP8, feel free to fix it. However, please make a dedicated commit.

If the content of a file does not respect the PEP8 and you want to modify it, please *stick to the PEP8*. Do not use the style of the file.

[1] https://github.com/jflesch/paperwork/issues

4.2 Commits

One commit = one change.

Try to make your commit message clear but concise.

When writing your commit message, please keep in mind the Git history is about the whole repository. Therefore, please give some context and indicate which part of Paperwork has been touched by your commit.

4.3 Comments

Too much is better than not enough. Other than that, there are no rules regarding comments. Please just use your common sense.

If your code is complex and not commented enough, it will be unmaintainable by others. It will then be *removed* as soon as you stop maintaining it. It already happened !

5 Paperwork itself

The code is split in two pieces:

- *backend* : Everything related to document and work directory management. May depend on various things but *not* GTK+
- *frontend* (aka *paperwork-gui* or .. *Paperwork*): The GUI. Entirely dependent on GTK+

The following describes the code organization. It is *not* an API documentation.

5.1 Paperwork-backend

See the user manual (section "advanced use") for a description of the work directory organization.

Paperwork-backend source code is available on GitHub[2].

5.1.1 docsearch.py

Contains two classes: DummyDocSearch and DocSearch. DummyDocSearch can just be used as temporary placeholder for DocSearch. DocSearch is the root class of the backend.

DocSearch manages everything related to the index. DocSearch provides:

- Access to the known documents in the index (methods get(), get_doc_from_docid(), find_documents()). All documents are instances of sub-classes of common/doc.py:BasicDoc.
- A way to guess labels on a document (method guess_labels())
- A way to get keyword suggestions (method find_suggestions())
- a DocDirExaminer instance. This object allows to scan a work directory to find the differences with what is known in the index.

[2] https://github.com/jflesch/paperwork-backend

- a DocIndexUpdater instance. This object allows to update the content of the index.
- Shortcuts for label management (methods create_label(), add_label(), remove_label(), update_label(), destroy_label())

When Paperwork starts, it uses the DocDirExaminer to find what has been changed. It keeps track of the changes. Once the scan is done, it transfer the list of documents to the DocIndexUpdater.

When a document is modified (labels, OCR re-done, etc), Paperwork uses the DocIndexUpdater to update the index.

5.1.2 Common to all documents

Files and directories structure Each document has its own folder.

In each of these folders, there is a file called ("labels"), containing the list of labels (and their color) applied to this document. Files "labels" are CSV files.

Folder names are arbitrary but by convention, they are dates with the following format: YYYYMMDD_hhmm_ss[_nn].

There may optionally be thumbnails for caching purpose : paper.<page_nb>.thumb.jpg.

Important: Files and directories structures may change in the future to solve various issues (atomic changes on labels, etc).

Programming All documents are instances of sub-classes of common/doc.py:BasicDoc.

All pages are instances of sub-classes of common/page.py:BasicPage.

5.1.3 Image documents

Image documents are usually created using a scanner. They can also be created by importing images (JPEG, PNG, etc).

Each image document is a folder. In this folder, there are image files (JPEG), one for each page.

- paper.<page_nb>.jpg : the scanned page
- paper.<page_nb>.words : the result of the OCR, in hOCR (HTML derivated) format (see the limitations of PyOCR for details)

5.1.4 PDF documents

Paperwork takes care of never modifying PDF documents. Modifying them could mean losing metadata (signatures, etc). So, when they are imported, they are simply copied.

Only one file is specific to PDF documents: doc.pdf.

5.1.5 docimport.py

Contains most of the code relative to document import. Provides various importers for various situations.

See docimport.IMPORTERS and docimport.get_possible_importers().

5.1.6 Export

Each document and each page provides methods for export:

- get_export_formats(): returns the possible formats in which the document/page can be exported
- build_exporter(): returns an exporter object: provides various export settings and preview

Additional exporter(s) is/are available in docexport.py. They are mostly used when exporting many documents in one shot.

5.2 Paperwork-shell

Paperwork-backend provides a command line utility: paperwork-shell.

Paperwork-shell provides various commands. Some are directly available with only paperwork-shell, some are coming from paperwork-gui.

5.3 Paperwork-gui

Paperwork-gui source code is available on GitHub[3].

6 Development tips

6.1 Configuration file for development only

Paperwork looks for a '*paperwork.conf*' in the current work directory before looking for a '*~/.config/.paperwork.conf*' in your home directory. So if you want to use a different configuration file and/or a different set of documents for development than for your real-life work, just copy your '*~/.config/paperwork.conf*' to '*./paperwork.conf*'. Note that the settings dialog will also take care of updating '*./paperwork.conf*' instead of '*~/.config/paperwork.conf*'.

6.1.1 Verbose

You can use the environment variable '*PAPERWORK_VERBOSE*' to increase or decrease the logging level. The accepted values are: *DEBUG*, *INFO*, *WARNING*, *ERROR*.

6.1.2 PEP8 and Pylint

There is a tool called '*pep8*' (*apt install pep8*). Use it.

Another tool is 'pylint'. A pylintrc is provided for Paperwork:

```
$ cd src ; pylint --rcfile=../pylintrc *
```

[3] https://github.com/jflesch/paperwork

7 Dependencies and constraints

7.1 GLib / GObject / GTK

GTK+ is not thread-safe.

7.2 Pillow

Thread-safe.

7.3 Whoosh

Thread-safe. Lock the GIL.

7.4 Poppler

Not thread-safe.

7.5 PyOCR

Thread-safe.

7.6 Pyinsane2, libsane, and scanner drivers

libsane is not thread-safe (mostly depending on the driver used). Previous versions of libsane + HP drivers were so sensitive that you had to call their functions always from the very same thread.

However, Pyinsane2 is thread-safe.

7.7 Libpillowfight / pypillowfight

Thread-safe.

8 Threads and processes

8.1 Thread safety

Thread safety is a major issue in Paperwork. We need threads to keep the GUI smooth, but unfortunately, not all Paperwork dependencies are thread-safe.

A job scheduling mechanism has been implemented (see *src/paperwork/frontend/util/jobs.py*). The idea here is to run most of the jobs in the same thread (whenever possible).

Each Job object represents a task to do. Some jobs can be stopped and resumed later (for instance JobDocThumbnailer). Each Job instance is used once and only once.

JobFactories instanciate Jobs. They are also used to keep the job recognizable.

Jobs are passed to JobSchedulers. Each JobScheduler represents a thread. They accept jobs using the method *schedule()*. The job with the higher priority is run first. If the job added to the scheduler has an higher priority than the active one, the scheduler will try to stop the active one and run it back later.

Jobs can be cancelled (assuming they are stoppable or not active yet). A single job can be canceled, or all the jobs from a given factory.

There is one main scheduler (called *main*). The main scheduler is the one used to access all the documents contents and the index.

Note that there are other threads running. For instance, there are the thread of PyInsane and the GTK+/GLib main loop.

8.2 Multi-processes

8.2.1 Pyinsane and Sane

In the past, some Sane drivers have shown to be unreliable. Some crashes, while some other corrupts memory areas sometimes.

To work around those issues, on GNU/Linux, Pyinsane forks (see *pyinsane2/sane/abstract_proc.py*). A dedicated process is used to scan. Two unix pipes are created for the communication.

8.2.2 Paperwork-backend and python-whoosh

Python-whoosh locks Python's GIL (Global Interpreter Lock). When the GIL is locked, the GUI is stuck.

As a workaround, paperwork-backend uses the *multiprocessing* module : It forks itself. One process acts as client (*docsearch.py*) and the other as server (*index.py*)

9 Most complex pieces

9.1 Scan process

9.2 OCR process

9.3 Index updates

9.4 Searching

10 Tests

11 Windows build

12 Versioning

Version have the following syntax: <M>[.<m>[.<U>[<i>][-<extra>]]]

- M = Major version : Major changes made / product completed.
- m = minor version : Minor changes made.
- U = update version : Only bug fixes
- i = installer revision : Only the installer has been changed (Windows only)
- Extra : (Git tags only) May match a Git tag done before a big change (for instance: before switching from Gtk 2 to Gtk 3). If extra == "git", indicates a version directly taken from the git repository.

13 Documentation

Everything related to documentation is in the directory doc/.

13.1 Installation

Documentation regarding the installation process is written using Markdown. This allows to show the documentation on GitHub

13.2 Other

Other part of the documentations are redacted using Lyx[4]. Lyx is a document processor with various advantages:

- Clearly structured documents
- .lyx files are ASCII files (can be versioned with Git easily)
- can be exported to PDF easily

Those documents are then packaged with Paperwork in PDF format. Paperwork can display them easily.

The document doc/intro_<lang>.lyx (fallback on intro.lyx if required) is the one automatically added and shown when there is no document in the work directory.

[4] https://www.lyx.org/