imag 0.8.0

User Documentation

April 2018

Contents

1 Introduction

This document is the user documentation for imag, the personal information management suite for the commandline. Besides being a documentation, it serves also as "roadmap" where this project should go.

Basically: This is Hobby stuff. Expect incompleteness, false statements and generally read with big grain of salt.

If you have any objections, suggestions for improvements, bugs, etc, please file them. A way to reach out to the imag project maintainer(s) is described in the CONTRIBUTING file of the repository or in this document, in the appropriate section.

1.1 The Problem

The problem this project tries to solve is to provide a modular commandline application for personal information management.

It targets "power users" or "commandline users", uses plain text as a storage format and tries to be scriptable. imag offers the ability to link data from different "PIM aspects" (such as "diary" and "bookmark" for example).

One major goal of imag is to make the PIM data traverseable and queryable. For example: a wiki article can be linked to an appointment which is linked to a todo which is linked to a note which is linked to a contact.

imag wants to offer an all-in-one scriptable modular commandline personal information management suite for all PIM aspects one can think of. Because imag uses plain text (TOML headers for structured data and plain text which can be rendered using markdown, for example, for continuous text) the user is always able to access their data without the imag tools at hand.

1.2 The Approach

The approach "imag" takes on solving this problem is to store content in a "store" and persisting content in a unified way. Meta-information is attached to the content which can be used to store structured data. This can be used to implement a variety of "domain modules" using the store. While content is stored in *one* place, imag does not duplicate content. imag does not copy or move icalendar files, emails, vcard files, music or movies to the store, but creates references to the actual files and stores meta-information in the store.

Detailed explanation on this approach follows in the chapters of this work.

1.3 Implementation

The program is written in the Rust programming language.

The program consists of libraries which can be re-used by other projects to implement and adapt imag functionality. An external program may use a library of the imag distribution to store content in the store of imag and make it visible to imag this way.

This is a technical detail a user does not necessarily need to know, but as imag is intended for power-users anyways, we could say it fits here.

1.4 Alternative Projects

imag is not the only project which tries to solve that particular problem. For example there is org mode for the emacs text editor. There is also zim, a desktop wiki editor which is intended to be used for a personal wiki.

The difference between imag and the mentioned projects is that imag is not there yet. Some parts can be used, though it is far away from being feature-complete.

2 Architecture of the imag code

The imag codebase has a rather simple overall architecture. In this chapter the types of crates, architecture of an imag module and the type structure are described.

2.1 Crate types

There are different types of crates in the imag world. A crate is a rust project.

First of all, there are core crates. These crates provide the very core of imag and almost all other crates use them:

- libimagstore - The imag store is the abstraction over the filesystem. It provides primitives to get, write and manipulate store entries and their header information.
- libimagrt - The runtime library, which provides functionality to create a store object from libimagstore, helps with configurarion loading and commandline argument handling (through the external "clap" crate).
- libimagerror - Error handling library for handling errors the imag way. Used in all other crates, even the store itself. It also offers functionality to log and trace errors as well as exiting the application, if necessary.
- libimagutil - Utilities.

The next type of imag crates are entry extension libraries. Those provide extensional functionality for the types from libimagstore. For example, there is "libimagentrylink" which provides functionality to link two entries in the store.

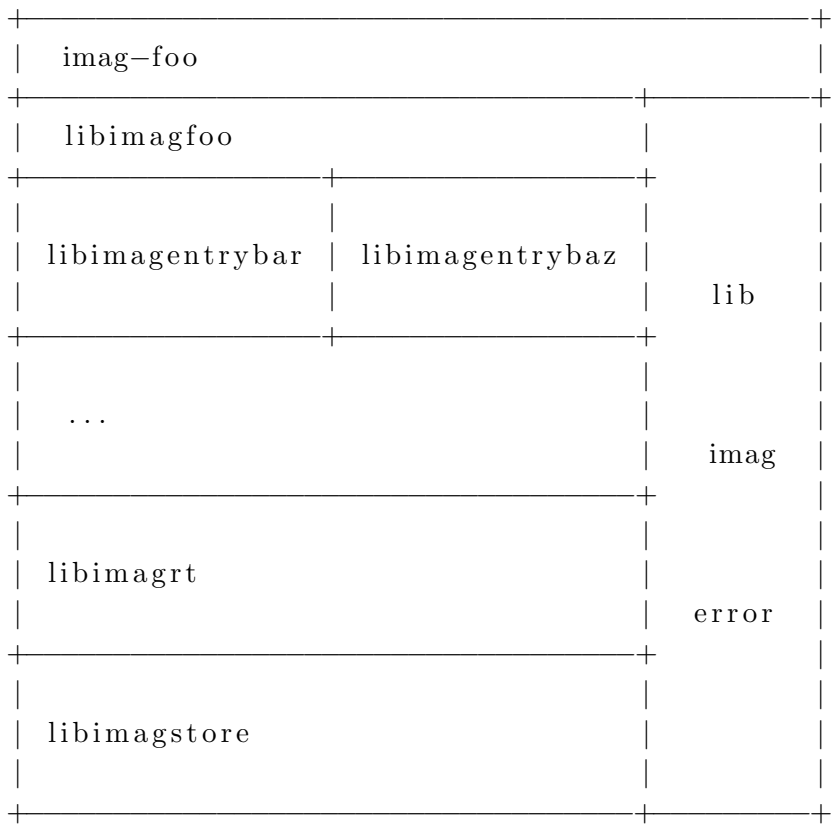
The third kind of crate is the one that offers end-user functionality for a imag domain, for example "libimagtodo" provides functionality to track todos.

And last, but not least, the commandline frontend crates provide the user interface. These are the kind of crates that are not library crates, but binaries.

Besides these, there are some other utility crates.

2.2 Architecture of an imag module

With the things from above, a module could have the following architecture:



The foundation of all imag modules is the store, as one can see in the visualization from above. Above the store library there is the libimagrt, which provides the basic runtime and access to the Store object. Cross-cutting, there is the error library (and possibly the util library, but we do not care about this one here), which is used through all levels. The highest level of all imag modules is the commandline interface on top of the domain library. In between can be any number of entry extension libraries, or none if not needed.

Theoretically, the commandline interface crate could be replaced to build a terminal user interface, graphical user interface or web interface.

2.3 Types

The imag core, hence the libimagstore, libimagrt and libimagerror, provide a set of types that a user (as in a library writer) should be aware of.

First of all, there is the Runtime type which is provided by the libimagrt. It provides basic access to whether debugging or verbosity is enabled as well as the most important core object: The Store.

The Store type is provided by the libimagstore library, the heart of everything.

When interacting with the store, two types are visible: FileLockEntry and Entry whereas the former derefs to the latter, which basically means that the former wraps the latter. The

FileLockEntry is a necessary wrapper for ensuring that when working concurrently with the store, an entry is only *borrowed* once from the store. It also ensures that the object is alive as long as the store is.

The Entry type provides functionality like reading the actual content, its header and so on. Extensions for its functionality are implemented on this type, not on the FileLockEntry.

The Entry provides access to its header, which is a toml::Value, where toml is the toml-rs crate (external project). Convenience functionality is provided via the toml−query crate, which is an external project which was initiated and extracted from the imag project.

Error types are also important. All errors in imag projects should be created with error−chain. libimagerror provides functionality to enhance the experience with Result types and general tracing of errors.

3 The Store

The store is where all the good things happen. The store is basically just a directory on the filesystem imag manages and keeps its state in.

One could say that the store is simply a database, and it really is. We opted to go for plain text, though, as we believe that plain text is the only sane way to do such a thing, especially because the amount of data which is to be expected in this domain is in the lower Megabytes range and even if it is *really* much won't exceed the Gigabytes ever.

Having a storage format which is plain-text based is the superior approach, as text editors will always be there.

A user should always be able to read her data without great effort and putting everything in a *real* database like sqlite or even postgresql would need a user to install additional software just to read his own data. We don't want that. Text is readable until the worlds end and we think it is therefore better to store the data in plain text.

The following sections describe the store and the file format we use to store data. One may skip the following sections, they are included for users who want to dig into the store with their editors.

3.1 File Format

The contents of the store are encoded in UTF-8. A normal text editor (like vim or the other one) will always be sufficient to dig into the store and modify files. For simple viewing even a pager (like less) is sufficient.

Each entry in the store consists of two parts:

1. Header
2. Content

The following section describe their purpose.

3.1.1 Header Format

The header format is where imag stores its data. The header is an area at the top of every file which is seperated from the content part by three dashes (−−−). Between these three dashes there is structured data. imag uses TOML as data format for this structured data, because it fits best and the available TOML parser for the rust programming language is really good.

The header can contain any amount of data, but modules (see Section 6) are restricted in their way of altering the data.

So normally there are several sections in the header. One section ([imag]) is always present. It contains a version field, which tells imag which version this file was created with.

Other sections are named like the modules which created them. Every module is allowed to store arbitrary data under its own section and a module may never read other sections than its own.

These conventions are not enforced by imag itself, though.

3.1.2 Content Format

The content is the part of the file where the user is free to enter any textual content. The content may be rendered as Markdown or other markup format for the users convenience. The store does never expect and specific markup and actually the markup implementation is not inside the very core of imag.

Technically it would be possible that the content part of a file is used to store binary data. We don't want this, though, as it is contrary to the goals of imag.

3.1.3 Example

An example for a file in the store follows.

```
---
[imag]
version = "0.8.0"

[note]
name = "foo"

[link]
internal = ["some/other/imag/entry"]
---

This is an example text, written by the user.
```

3.2 File organization

The "Entries" are stored as files in the "Store", which is a directory the user has access to. The store may exist in the users Home-directory or any other directory the user has read-write-access to.

Each module stores its data in an own subdirectory in the store. This is because we like to keep things ordered and clean, not because it is technically necessary.

We name the path to a file in the store "Store id" or "Storepath" and we often refer to it by using the store location as root. So if the store exists in /home/user/store/, a file with the storepath /example.file is (on the filesystem) located at /home/user/store/example.file.

By convention, each libimagentry<name> and libimag<name> module stores its entries in in /<name>/.

So, the pattern for the storepath is

```
/<module name>/<optional sub-folders>/<file name>
```

Any number of subdirectories may be used, so creating folder hierarchies is possible and valid. A file "example" for a module "module" could be stored in sub-folders like this:

```
/module/some/sub/folder/example
```

3.3 Backends

The store itself also has a backend. This backend is the "filesystem abstraction" code.

Note: This is a very core thing. Casual users might want to skip this section.

3.3.1 Problem

First, we had a compiletime backend for the store. This means that the actual filesystem operations were compiled into the store either as real filesystem operations (in a normal debug or release build) but as a in-memory variant in the 'test' case. So tests did not hit the filesystem when running. This gave us us the possibility to run tests concurrently with multiple stores that did not interfere with each other.

This approach worked perfectly well until we started to test not the store itself but crates that depend on the store implementation. When running tests in a crate that depends on the store, the store itself was compiled with the filesystem-hitting-backend. This was problematic, as tests could not be implemented without hitting the filesystem and mess up other currently-running tests.

Hence we implemented store backends.

3.3.2 Implementation

The filesystem is abstracted via a trait FileAbstraction which contains the essential functions for working with the filesystem.

Two implementations are provided in the code:

- FSFileAbstraction
- InMemoryFileAbstraction

whereas the first actually works with the filesystem and the latter works with an in-memory HashMap that is used as filesystem.

Further, the trait FileAbstractionInstance was introduced for functions which are executed on actual instances of content from the filesystem, which was previousely tied into the general abstraction mechanism.

So, the FileAbstraction trait is for working with the filesystem, the FileAbstractionInstance trait is for working with instances of content from the filesystem (speak: actual Files).

In case of the FSFileAbstractionInstance, which is the implementation of the FileAbstractionInstance for the actual filesystem-hitting code, the underlying resource is managed like with the old code before. The InMemoryFileAbstractionInstance implementation is corrosponding to the InMemoryFileAbstraction implementation - for the in-memory "filesystem".

3.4 The StdIo backend

Sidenote: The name is "StdIo" because its main purpose is Stdin/Stdio, but it is abstracted over Read/Write actually, so it is also possible to use this backend in other ways, too.

3.4.1 Why?

This is a backend for the imag store which is created from stdin, by piping contents into the store (via JSON or TOML) and piping the store contents (as JSON or TOML) to stdout when the backend is destructed.

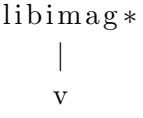
This is one of some components which make command-chaining in imag possible. With this, the application does not have to know whether the store actually lives on the filesystem or just "in memory".

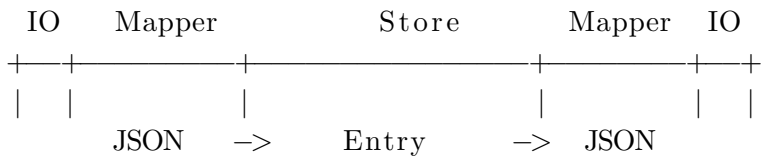
3.4.2 Mappers

The backend contains a "Mapper" which defines how the contents get mapped into the in-memory store representation: A JSON implementation or a TOML implementation are possible.

The following section assumes a JSON mapper.

The mapper reads the JSON, parses it and translates it to a Entry. Then, the entry is made available to the store codebase. To summarize what we do right now, lets have a look at the awesome ascii-art below:





This is what gets translated where for one imag call with a stdio store backend.

3.4.3 The JSON Mapper

The JSON mapper maps JSON which is read from a source into a HashMap which represents the in-memory filesystem.

The strucure is as follows:

```
{
    "version": "0.8.0",
    "store": {
        "example": {
            "header": {
                "imag": {
                    "version": "0.8.0",
                },
            },
            "content": "hi there!",
        },
    },
}
```

3.4.4 TODO

If you look at the version history of this file you will see that this implementation has grown from something complex and probably slow to what we have today.

Still, there's one improvement we could make: abstract all the things away so the libimag∗ crates handle the header without knowing whether it is JSON or TOML. With this, we would not even have to translate JSON to TOML anymore. We should measure whether this would have actually any performance impact before implementing it.

4 Conventions, best practices

This section explains conventions used in the imag codebase. It is mainly focused on developers, but a user may read it for getting to know how imag works.

Lets work our way up from the store and how to extend it to the commandline user interface.

4.1 Versioning

All imag crates are versioned with the same version number until we reach some "1.0.0" version. This means that all imag tools are only tested for compatibility with libraries and such if their version numbers match. It might not be possible to import one imag library in version 0.3.0 and another one in 0.4.0 and make them work together. It also means that if new tools are introduced into the imag codebase, they might start with their first version not at 0.1.0 but at something like 0.5.0.

4.2 Store and Entry functionality

A Entry does not offer much functionality by itself. So its the job of libraries to *extend* its functionality. This should never be done by wrapping the Entry type itself but by providing and implementing an extension trait on it.

Same goes for extending the Store type: never wrap it, always provide an extension trait for it.

These two rules ensure that the type does not lose any functionality from a wrapping. Deref could do that, but not over muliple levels, so extension traits it is. It also most likely results in functions inside the extension trait which all return a Result<_, _>.

4.3 Libraries

In the next few sections, conventions and best practices for writing a imag library are written down.

A developer of imag should read this carefully, a user may skip this section or cross-read it for better understanding of the imag project.

4.3.1 Library naming

Libraries which provide functionality for entries or the store (most likely entries or both) should be named "libimagentrything" whereas "thing" stands for what the library provides.

All other libraries should be prefixed with "libimag" at least. Most likely, one will not write such a library but rather a "libimagentrything" library.

4.3.2 Library scope

A library should never introduce utility functionality which could be useful for other libraries as well. If there is no such functionality available, the "libimagutil" or "libimagentryutil" might be a place where such a function would be put.

If a library has to introduce free functions in its public interface, one should think hard whether this is really necessary.

4.3.3 Library error types/kinds

Libraries must use "error-chain" to create error types and kinds. Most likely, a library needs some kinds for wrapping the errors from underlying libraries, such as the store itself.

A library must *never* introduce multiple error types, but is free to introduce as many error kinds as required.

4.3.4 Libraries with commandline frontends

Libraries with commandline frontends provide end-user functionality. Normally, they depend on one or more "libimagentrything" libraries. They should be named "libimagthing", though. For example: "libimagdiary", "libimagtimetrack" or "libimagwiki", whereas the commandline frontends would be "imag-diary", "imag-timetrack" and "imag-wiki", respectively.

If such a library needs to depend on another "libimagthing", for example if "libimagdiary" needs to depend on "libimagnote", one should think about this and whether the functionality could be outsourced to a more general "libimagentrything".

A library which implements a functionality for imag may contain helper functions for commandline stuff, but that is discouraged.

4.3.5 Library testing

All libraries should be tested as much as possible. Sometimes it may not be possible without a lot of effort, but still: more tests = better!

4.4 Commandline tools

The commandline tools are the CLI-frontends for their respective libraries. So libimagdiary has a CLI frontend imag−diary.

Those CLI frontends use functionality from libimagrt to build a consistent commandline interface.

5 Writing an imag module

So you want to write a module for imag. That's nice.

This guide helps you getting started. It also can help you understanding how imag modules work, so even if you do not want to write a full new module, but extend or alter one, this guide may help you.

5.1 Data layout

First, you have to think about what data you want to store. What functionality do you want to provide and what data that creates.

In this example, we're writing a module that stores numbers. We're writing the appropriate library for that as well as a commandline frontend.

5.2 libimagnumberstorage

We're writing a libimagnumberstorage which provides the core functionality of our module: Storing numbers.

That library can then be used by other library authors and by the commandline interface implementation.

5.2.1 Setup

So what do we need to do to write this library:

1. Create a new "lib" crate. Because we're writing a "domain" library, we're doing this in the lib/domain subdirectory: cd lib/domain; cargo new --lib libimagnumberstorage.
2. After creating the library, we have to add the new library to the /Cargo.toml field and add the missing metadata in the new /lib/domain/libimagnumberstorage/Cargo.toml file.

That was the setup part. Now we can implement our functionality. For that, we need to *extend* two types from libimagstore, so we have our first dependency here.

5.2.2 Dependencies to other libraries

3. Put libimagstore as a dependency in the /lib/domain/libimagnumberstorage/Cargo.toml file. By using libimagstore = { version = "0.8.0", path = "../../../ lib/core/libimagstore" } we automatically get all the goodness of Cargo, so that releases automagically work as expected, but when developing locally, the local version of libimagstore is used. Of course, the version has to be the latest released version.
4. For error handling, we also need to import libimagerror.
5. For easy header-editing, we import toml and toml-query.
6. For error-type creating, we import error-chain.

5.2.3 Interface

7. Then, we have to *extend* two types:
    1. libimagstore::store::Store has to be extended so we can implement a CRUD interface for our special entries.
    2. libimagstore::store::Entry has to be extended so we can get our stored numbers in a convenient way.

Our interface should roughly look like this:

```
store.get_stored_number("5") -> Result<FileLockEntry, _>
store.store_number("5")      -> Result<FileLockEntry, _>
store.delete_number("5")     -> Result<(), _>
```

You notice that the Store returns FileLockEntry objects rather than Entry objects. And that's ok. A FileLockEntry is a Entry, but ensures that we are the only ones editing that entry. So, we have to implement our number-storing-interface on Entry as well:

```
entry.get_number() -> Result<usize>
entry.set_number(usize) -> Result<()>
```

All those "extensions" are implemented as traits which are then implemented for Store and Entry.

Normally, we create new files for that, as well as for the error types we need:

- /lib/domain/libimagnumberstorage/src/store.rs
- /lib/domain/libimagnumberstorage/src/entry.rs
- /lib/domain/libimagnumberstorage/src/error.rs

where store.rs contains a trait NumberStorage and entry.rs contains a trait NumberEntry. error.rs contains the invocation of the error_chain!{} macro. Error types from libimagstore and others are linked in.

6 Modules

A module is a functionality of the program. There is a huge list of modules available in the imag core distribution.

Some of the modules shipped with imag cover core functionality such as linking, tagging or references to files outside of the store or even the store interface itself. Others cover things like diary, notes, wiki or bookmarks. These are also called "domains".

We try really hard to offer a consistent commandline user interface over all of these modules.

The following sections describe each module in detail, including its purpose and its provided functionality.

6.1 Bookmarks

The Bookmarks module is for keeping URLs as bookmarks, tagging and categorizing them and finally also open them in the browser.

6.2 Diary

The diary module is for keeping your diary notes.

The diary module giv3s you the possibility to write your diary in imag. It offers daily, hourly and minutely entries (the latter beeing more like a private tumble-blog).

Exporting the diary is possible, so one can write it in markdown and later pass that to pandoc, if desired, to generate a website or book from it.

6.3 Edit

The imag−edit command is for simply editing store entries with the $EDITOR.

It is based on libimagentryedit (Section 7.7).

6.4 Init

This is the only imag−∗ command which does *not* set up a runtime and check whether the store is available. This command can be used to set up a imag store.

It also puts a default configuration in the right place and initializes a git repository, if there is a git command in $PATH (via calling git on the commandline, not via libgit2 or some other library).

6.5 Link

The linking module imag−link is one of the plumbing modules. It offers the possibility to link entries in the store.

It also offers the functionality to link to external sources. This functionality *can* be used to link to external URLs, but the bookmarking module should be used to do this (see Section 6.1).

The linking module offers functionality to add, remove and list both internal (store entry to store entry) and external (store entry to URL) links.

6.5.1 Internal linking

6.5.2 External linking

A store entry can only have *one* external link. Therefor, when you create an external link, the linking module creates a new entry in the store which links to this URL. The linking module then links you entry with this new entry by using an internal link. This way one entry can have multiple external links attached to it and external links are deduplicated automatically.

6.6 Log

The “imag-log” module is a lightweight interface to the “imag-diary” command.

It is intended as a tumbeblog-like diary, where one does not care to fire up an editor and type in a long text, but rather type a few words and forget about it:

6.6.1 Usage

Logs can be created via an entry in the configuration file in the section log:

```
[log]
logs = ["work", "hobby", "music"]
default = "hobby"
```

The default key is required and the name which is used here *must* appear in the logs array.

In the above configuration snippet, the logs work, hobby and music are created. The user may now log to one of these logs with:

```
imag log --to <logname> "Some message"
# or
imag log -t <logname> "Some message"
# or, to the default log:
imag log "Some message"
```

Logs can be read by naming the log:

```
imag log show work
```

which prints one log per line (including time it was logged).

6.7 Mails

The Mails module implements a commandline email client. Emails can be written (via $EDITOR) and viewed, also in threads. Emails can be crawled for creating new contacts.

A Text User Interface is not planned, but might be there at some point.

The mail module implements a minimal Email client. It does not handle IMAP syncing or SMTP things, it is just a *viewer* for emails (a MUA).

The goal of the initial implementation is only a CLI, not a TUI like mutt offers, for example (but that might be implemented later). As this is an imag module, it also creates references to mails inside the imag store which can be used by other tools then (for example imag-link to link an entry with a mail - or the imag entry representing that mail).

So this module offers functionality to read (Maildir) mailboxes, search for and list mails and mail-threads and reply to mails (by spawning the $EDITOR).

Outgoing mails are pushed to a special directory and can later on be send via imag-mail which calls a MTA (for example msmtp) and also creates store entries for the outgoing mails.

6.7.1 CLI

The CLI of the imag-mail module is planned as follows:

```
imag mail track <path> [opts...]  # track a new mail, mail file passed as p
imag mail scan <path> [opts...]   # scan a maildir and track all untracked
```

```
imag mail box <name|path>          # work with the mailbox specified by <nam
imag mail list <args...>           # list mails in a given mailbox for a giv
imag mail show <args...>           # open new mails in the pager
imag mail thread list <args...>    # list mails from a thread
imag mail thread show <args...>    # open new mails from a thread in the pag
imag mail new <args...>            # craft a new mail and safe it in the <ou
imag mail send <args...>           # send emails from the outgoing folder, o
imag mail mv <srcbox> <dstbox>     # move a mail (or thread) from one mailbo
```

6.8 Notes

The Notes module is intended to keep notes. These notes can be inserted as plain text, markdown or other markup languages.

The notes module offers:

- adding, removing and settings of tags
- listing notes, optionally filtered by
  - tags
  - grepping through note content and listing
    - the matches
    - files with matches
- opening a note via xdg−open (rendered as HTML if content is written in a markup language)

6.9 Reference

The Reference module.

6.10 Store

The Store module.

6.11 Tagging

The Tagging module.

A valid tag matches the regex [a−zA−Z][0−9a−zA−Z]∗.

6.12 Timetrack

The Timetrack module implements a timewarrior-like timetracking functionality for imag.

Each timetracking is a ‘tag’ which can be started and stopped. These tags are *no* tags as in imag-tag, but timetracking-tags.

Summaries can be printed, also filtered by tags if desired.

6.13 Todo

The Todo module implements taskwarrior functionality by integrating taskwarrior itself into imag.

Each taskwarrior task s referenced from imag and represented as imag entry, thus making it linkable by other imag entries.

6.14 View

The View module.

6.15 Wiki

The Wiki module provides a personal wiki implementation.

The wiki entries are markdown-formatted files in the imag store. All entries are automatically searched for links and those links are automatically added to the header (or as external link, depending on the format).

Wiki entries can have no or one category and a arbitrary number of tags. Entries can be listed (as a "tree" shape) and filtered by content, category and tag.

7 Libraries

This section of the documentation is only relevant for developers and you might skip it if you're only a user of the imag tool.

The following sections contain a short documentation on what the several libraries are supposed to do. It is generated from the README.md files of each library and only gives a general overview what can be done with the library. For a more comprehensive documentation of the library, one might consult the appropriate documentation generated from the source of the library itself.

The documentation of the libraries is sorted *alphabetically*.

7.1 libimagbookmark

This library crate implements functionality for bookmarks.

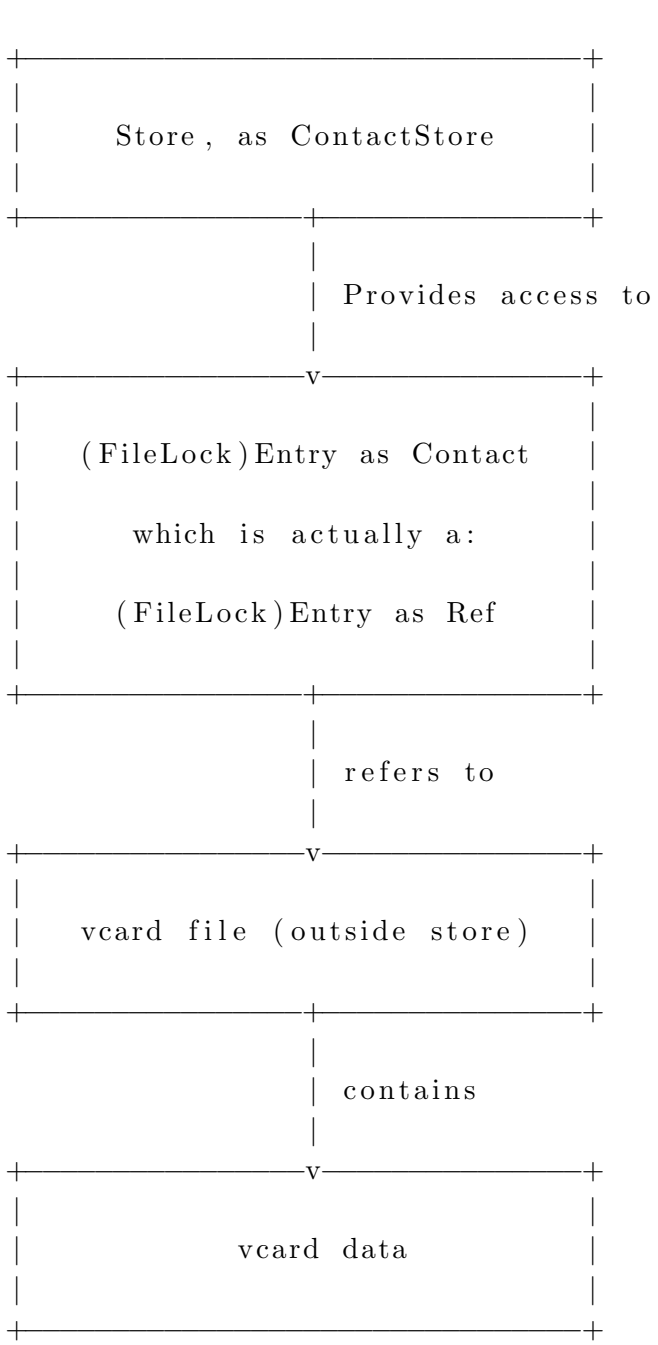
It uses libimagentrylink to create external links and therefor deduplicates equivalent external links (libimagentrylink deduplicates - you cannot store two different store entries for https://imag−pim.org in the store).

It supports bookmark collections and all basic functionality that one might need.

7.2 libimagcontacts

The contact library basically only creates references to the actual vcard files, though it also can parse (via the vobject crate) the information and return it from an entry directly.

The architecture of indirections is as follows:



As the library is build upon libimagentryref, it does not create a new subcollection in the store /contacts, but uses the infrastructure of libimagentryref which automatically puts all references in /ref.

7.3 libimagdiary

This library crates implements a full diary.

One can have one or more diaries in the store, each diary can have unlimited entries.

7.3.1 Future plans

The diary should be able to provide *daily*, *hourly* and even *minutely* diary entries, so one can use the diary as normal "Dear diary, today…"-diary, or more fine-grained and more like a journal.

The internal file format as well as the store-path generation for this module is prepared for such functionality.

7.4 libimagentryannotation

This library provides annotation functionality for entries.

Annotations are normal Store entries, but their header at annotation.is_annotation is set to true.

Annotations are linked to an entry (as in libimagentrylink).

7.4.1 Library functionality

The library features two traits: One to extend an Entry with annotation functionality and another one for extending the Store with functionality to get annotations of an entry and all annotations in the store.

7.5 libimagentrycategory

This library provides category functionality for entries.

7.6 libimagentrydatetime

Provides date/time functionality for entries.

7.7 libimagentryedit

Provides edit (as in spawning an $EDITOR) functionality for entries.

7.8 libimagentryfilter

Helper library to filter lists of entries by certain predicated. Offers filters for filtering by header values and other predicates, plus this library offers logical operants to combine filters.

A commandline-to-filter DSL is planned for this, so commandline applications can use this to implement a uniform filter interface.

7.9 libimagentrylink

Linking library for linking entries with other entries. Used for "imag-link", the commandline utility, but intended for use in other binaries and libraries as well.

7.10 libimagentrymarkdown

Helper crate to add useful functionality in a wrapper around hoedown for imag.

Adds functionality to extract links, parse content into HTML and other things which might be useful for markdown rendering in imag.

7.11 libimagentryref

This library crate contains functionality to generate *references* within the imag store.

A reference is a "pointer" to a file or directory on the filesystem and outside the store. It differs from libimagentrylink/external linking as it is designed exclusively for filesystem references, not for URLs.

A reference is created with a unique identifier, like a hash. The implementation how this hash is calculated can be defined by the user of libimagentryref.

So this library helps to resemble something like a *symlink*.

7.11.1 Usage

Users have to implement the UniqueRefPathGenerator trait which should implement a hashing functionality for pathes.

7.11.2 Limits

This is *not* intended to be a version control system or something like that. We also can not use *real symlinks* as we need imag-store-objects to be able to link stuff.

7.11.3 Usecase

This library offers functionality to refer to content outside of the store. It can be used to refer to *nearly static stuff* pretty easily - think of a Maildir - you add new mails by fetching them, but you mostly do not remove mails. If mails get moved, they can be re-found via their hash, because Maildir objects hardly change. Or because the hash implementation which is used to refer to them hashes only the Message−Id and that does not change.

7.11.4 Long-term TODO

Not implemented yet:

- [ ] Re-finding of files via their hash. This must be implemented with several things in mind * The user of the library should be able to provide a way how the filesystem is searched. Basically a Functor which yields pathes to check based on the original path of the missing file. This enables implementations which do only search a certain subset of pathes, or does depth-first-search rather than breadth-first-search.

7.12 libimagentrytag

Library for tagging entries. Used in "imag-tag" but should be used in all other modules which contain tagging functionality, so the backend and frontend look the same for all modules.

7.13 libimagentryutil

This library contains utilities for working with libimagstore :: store :: Entry objects where the functionality does not necessarily belong into libimagstore.

7.14 libimagentryview

Provides viewing (as in piping to stdout, opening in $EDITOR or in $BROWSER) functionality for entries.

7.15 libimagerror

In imag, we do not panic.

Whatever we do, if we fail as hard as possible, the end-user should *never ever* see a backtrace from a panic!().

Anyways, the user *might* see a error trace generated by imag. That is because imag is software for power-users, for nerds (I use the term "nerd" because for me it is a good thing - I do not want to offend anyone by using it). This target group can read backtraces without getting confused. IO Error and Permission denied Error are things that nerds can understand and they already know what to do in the most obvious cases (such as Permission denied Error).

This library crate is for generating error types and handle them in a nice way. It can be seen as mini-framework inside imag which was written to work with error types in a specified way. All imag crates *must* use this library if they can return errors in any way, except the libimagutil - which is for the most basic utilities.

7.16 libimaghabit

The habit library implements a habit tracker.

A habit can be instantiated with a name and a time-period in which it should be fullfilled (eg. daily, ever 3 days, weekly...).

The module offers ways to generate statistics about habits.

7.17 libimaginteraction

A crate for more general interaction with the user (interactive commandline interface).

Offers functions for asking the user Y/N questions, for (numeric) values, etc.

7.18 libimaglog

A small extension over libimagdiary which strips down the functionality of libimagdiary to some defaults for writing a log (a tumbleblog like diary) with rather short messages.

Provides only basic functionality over libimagdiary, most notably the "log.is_log" header entry, so the imag−log CLI can distinguish between "logs" and "diary entries".

7.19 libimagmails

The mail library implements everything that is needed for beeing used to implement a mail reader (MUA).

It therefor providea reading mailboxes, getting related content or mails, saving attachements to external locations, crafting new mails and responses,...

It also offers, natively, ways to search for mails (which are represented as imag entries) via tags, categories or even other metadata.

For more information on the domain of the imag−mail command, look at the documentation of the Section 6.7 module.

7.20 libimagnotes

7.21 libimagrt

This library provides utility functionality for the modules and the binary frontends, such as reading and parsing the configuration file, a builder helper for the commandline interface and such.

It also contains the store object and creates it from configuration.

the libimagrt :: runtime::Runtime object is the first complex object that comes to live in a imag binary.

7.21.1 Long-term TODO

- [ ] Merge with libimagstore

7.22 libimagstore

The store is the heart of everything. Here lives the data, the complexity and the performance bottleneck.

The store offeres read/write access to all entries.

The store itself does not offer functionality, but has a commandline interface "imag-store" which can do basic things with the store.

7.22.1 Long-term TODO

- [ ] Merge with libimagrt

7.23 libimagtimetrack

A library for tracking time events in the imag store.

7.23.1 Store format

Events are stored with a store id like this:

```
/timetrack/<insert-date-year>/<insert-date-month>/<insert-date-day>/<insert
```

Timetrackings contain

- a comment (optional, free text)
- a start date
- an end date
- a tag

by default and might be extended with more header fields as one likes.

The header of a timetrack "work" entry looks like this:

```
[event]
tag = "work"
start = "2017-01-02T03:04:05"
end = "2017-01-02T06:07:08"
```

Normal tags (as in libimagentrytag) are explicitely *not* used for tagging, so the user has the possibility to use normal tags on these entries as well.

The tag field is of type string, as for one tag, one entry is created. This way, one can track overlapping tags, as in:

```
imag timetrack start foo
imag timetrack start bar
imag timetrack stop foo
imag timetrack start baz
imag timetrack stop bar
imag timetrack stop baz
```

The end field is, of course, only set if the event already ended.

7.23.2 Library functionality

The library uses the libimagentrydatetime::datepath::DatePathBuilder for building StoreId objects.

The library offers two central traits:

- TimeTrackStore, which extends a Store object with functionality to create FileLockEntry objects with a certain setting that is used for time-tracking, and
- TimeTracking, which extends Entry with helper functions to query the entry-metadata that is used for the time tracking functionality

The library does *not* provide functionality to implement imag−timetrack or so, as the core functionality is already given and the commandline application can implement the missing bits in few lines of code.

Aggregating functionality might be provided at a later point in time.

7.24 libimagtodo

The library for the todo module.

Whether this wraps taskwarrior or implements a todo tracking mechanism in imag itself is to be defined. Probably the latter.

7.25 libimagutil

Utility library. Does not depend on other imag crates.

7.26 libimagwiki

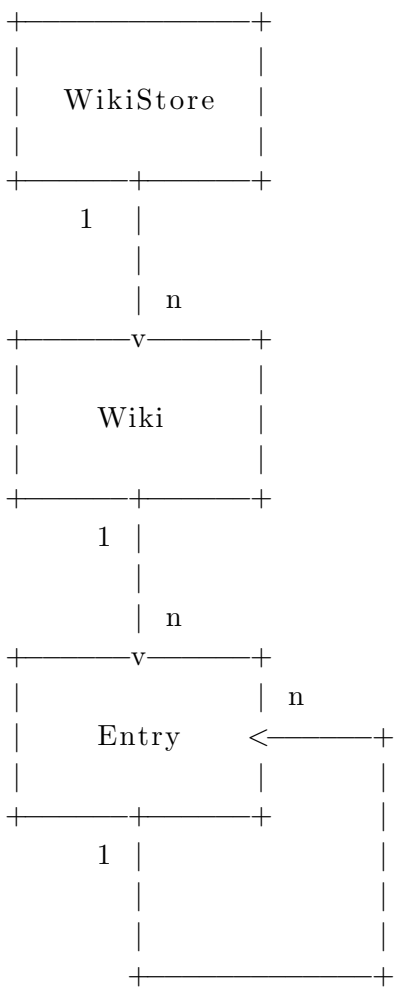
The wiki library implements a complete wiki for personal use.

This basically is a note-taking functionality combined with linking.

7.26.1 Layout

The basic structure and layout is as simple as it gets:

/wiki holds all wikis. The default wiki is /wiki/default. Below that there are entries. Entries can be in sub-collections, so /wiki/default/cars/mustang could be an entry.



The store offers an interface to get a Wiki. The wiki offers an interface to get entries from it.

Each Entry might link to a number of other entries *within the same wiki*. Cross-linking from one wiki entry to an entry of another wiki is technically possible, but not supported by the Entry itself (also read below).

When creating a new wiki, the main page is automatically created.

7.26.2 Autolinking

The Entry structure offers an interface which can be used to automatically detect links in the markdown. The links are then automatically linked (as in libimagentrylink).

8 Contributing to imag

So you want to contribute to imag! Thank you, that's awesome!

All contributors agree to the developer certificate of origin by contributing to imag.

8.1 Without Github

Contributing without a github account is perfectly fine and actually encouraged as we try to move away from github step by step. Feel free to contact us via our mailinglist and/or submit patches via mail (use git format−patch and git send−email, always add a cover letter to describe your submission).

Also ensure that each commit submitted via email has a "Signed-off-by:" line. By adding that line, you agree to our developer certificate of origin. If you do not add the "Signed-off-by:" line, I reserve the right to kindly reject your patch.

Once *I am* okay with your patchset, I will submit it as PR in the github repository (as long as we're using github), so CI can test it. I might come back to you if something broke in CI or someone has a suggestion how to improve your PR. I will keep you as author of the commits.

8.2 Finding an issue

Finding an issue is simple: We have a special label in our issues section for easy-to-solve issues. You can start there, don't hesitate to ask questions if you do not understand the issue comment! If there are currently no issues with that tag, just browse the issues or the code... you'll always find things to improve!

Also, if you've found bugs or outdated stuff in our documentation, feel free to file issues about them or even better: Write a pull request to fix them!

8.3 Prerequisites

The prerequisites are simple: cargo and rustc in current version (stable) or newer (we do not use nighly features though).

Build dependencies for building are listed in the default.nix file, though you do not have to have the nix package manager installed to build imag. Everything else will be done by cargo.

Note that this software is targeted towards commandline linux users and we do not aim to be portable to Windows or Mac OSX (though I wouldn't mind merging patches for OS X compatibility).

If you want to build the documentation (you don't have to) you'll need:

- pandoc
- pandoc-citeproc
- texlive
- lmodern (font package)
- (gnu) make

All dependencies are installable with the nix package manager by using a nix−shell, if you have the nix package manager installed on your system.

8.4 Commit guidelines

Please don't refer to issues or PRs from inside a commit message, if possible. Make sure your PR does not contain "Fixup" commits when publishing it, but feel free to push "Fixup" commits in the review process. We will ask you to clean your history before merging! If you're submitting via patch-mail, I will do the fixup squashing myself. If it fails I will come back to you.

Make sure to prefix your commits with "doc: " if you change the documentation. Do not change document and code in one commit, always separate them.

If your changes are user-visible (new commandline flags, other semantics in the commandline, etc), make sure to add a note in the CHANGELOG.md file (in the same commit if it is a simple change). **If it is a bugfix**, do add the changelog entry in a new commit (best would be: one commit for a testcase which shows the bug, one commit for the fix, more if the fix is complicated, and one commit for the changelog entry). Changelog entries for bug fixes should be extra commits, because backporting bugfixes gets simpler this way.

We do not follow some official Rust styleguide for our codebase, but we try to write minimal and readable code. 100 characters per line, as few lines as possible, avoid noise in the codebase, ... you get it.

Not all of your commits have to be buildable. But your PR has to be before it will be merged to master.

8.5 Feature branches

Use feature branches. If you could name them "/", for example "libimagstore/add-debugging-calls", that would be awesome.

8.6 Code of Conduct

We use the same code of conduct as the rust community does.

Basically: Be kind, encourage others to ask questions - you are encouraged to ask questions as well!

8.7 Developer Certificate of Origin

```
Developer Certificate of Origin
Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.
660 York Street, Suite 102,
San Francisco, CA 94110 USA
```

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or

(b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or

(c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.

(d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

9 Changelog

This section contains the changelog.

We try to include a changelog line in each pull request, to make sure the changelog is up to date when releasing a new version of the codebase. Make sure to append the new change to the list, do not prepend it.

The "Major" section of each section includes huge changes in functionality and interfaces (but not necessarily user-facing ones), whereas the "Minor" section contains only small stuff. Some things, like typo fixes, version string updates and such are not stated in the changelog (though updating of dependencies is). Please note that we do not have a "Breaking changes" section as we are in Version 0.y.z and thus we can break the API like we want and need to.

9.1 Next

This section contains the changelog from the last release to the next release.

- Major changes
- Minor changes
- Bugfixes

9.2 0.7.0

- Major changes
  - imag−timetrack list −−from/−−to now have kairos support - that means that complex −−from/−−to arguments (like yesterday or today−2weeks) are now possible
  - libimagerror got a major refactoring and uses ChainedError from error−chain for logging errors now.
  - libimagentryref and all libraries using it were rewritten. libimagentryref was rewritten to make its API simpler and yet more powerful. Also because it used to put entries under a "ref" collection in the store, but users of the library really should be be able to put entries under custom collections.
  - imag store ids was replaced by imag ids.
  - libimagentrylist was removed. Its functionality was inconvenient to use and ugly to implement. Its API was cumbersome. Listing of entries shall be implemented without it.
  - libimagcontact is now able to fetch all contacts from the store.
  - libimagcontact takes the hash from the vcard object (UID) now.
  - imag−contact got a find command, which matches in fullname, email and address and either shows or lists the found contacts
  - imag−contact list and imag−contact find is now able to print the output as JSON.
  - imag−edit can now read store ids from stdin, so imag ids | fzf | imag edit −I is now a thing.
  - imag ids does not print the path of the store. Can be turned on using commandline flag.
  - imag−habit today −−done and imag−habit status −−done was added for showing habits which are already done.
  - libimagrt allows external subcommands now in the default clap app builder helper. It also provides a helper for handling unknown subcommands: Runtime::handle_unknown_su
    See docs for details.
  - imag−link list prints output in ascii-table now, use −−plain to print as plain text.
  - The build script automatically generates autocomplete scripts for bash, fish and zsh now when compiling the imag command.
  - libimagwiki and imag−wiki were introduced.
- Minor changes

  - A license-checker was included into the CI setup, which checks whether all ".rs"-files have the license header at the top of the file
  - imag−link does not allow linking the entry to itself
  - imag sorts available commands alphabetically now
  - imag has a new subcommand help for consistency with other tools
  - imag−grep does not print grep statistics when only files with matches are listed
  - The "Ok" output which was printed on success was removed from all commands
  - imag−log show was aliased to imag−log list
  - imag−∗ −−version shows git describe output if binary was compiled in "debug" mode.
  - imag−diary supports "daily" diaries now.
  - imag−contact joins multiple emails with "," now
  - imag−tag commandline was rewritten for positional arguments.
  - libimagrt automatically takes "rt.editor" into account when building editor object
  - libimagentryref got a utility function for making an entry a ref.
  - libimaghabit got Habit:: instance_exists_for_date ()
  - imag contact find understands −−format now.
  - imag contact uses "," as seperator for output of lists of values.
  - imag contact find −−id / −−full−id was added for printing Store Id / Filepath of found contacts.
  - imag view can now view multiple entries at once
  - imag view −I reads store ids from stdin now.
  - libimagstore iterators have less restricting lifetimes now
  - libimagentrygrep was introduced, a crate for seaching in the header/content parts of an entry.
  - imag−ids can now filter by collection
  - All crates use "clap" with the "wrap_help" feature now.

- Bugfixes

  - imag does not panic anymore when piping and breaking that pipe, for example like with imag store ids | head −n 1. For that, libimagerror got a Result extension which can translate errors into exit codes and one for unwrapping or exiting with the Err(i32) from the result.
  - libimagdiary did not add the header markers on diary entries.
  - imag−diary used the default diary rather than the CLI setting. Now it rather uses the CLI setting and only if that is not present, it uses the default.
  - libimagerror printed errors with write!() rather than writeln!() when tracing.
  - A parsing error in libimagstore, which caused the parsing of entries with a line "—" in the content part to fail, was fixed.
  - The patch explained by the point above introduced a bug which caused entries to be read as a single line, which was fixed as well.
  - imag−diary create −−timed did not work as expected
  - libimagstore got another fix with the file parsing, as the std :: str :: Lines iterator takes empty lines as no lines.
  - libimagentryedit fixed to inherit stdin and stderr to child process for editor com-

mand.
- libimagrt produced the editor command without taking arguments into account.
- libimagentryref got a fix where the buffer for the hash calculation was not allocated properly.
- libimagstore :: store :: Store :: create overwrote existing entries.
- libimaghabit :: habit :: HabitTemplate did not link new instances.
- imag−init creates ~/.imag but not ~/.imag/store.
- libimagrt got a bugfix in the editor command setup where command arguments were not processed correctly which could result in calling the editor with an empty argument (vim " ").
- imag−grep did not count in all cases.
- libimagdiary sorts entries by date when viewing/listing them.
- A libimagentryref bug was fixed where the wrong variable was passed as path to the referenced file, causing all tools based on this lib to break.
- libimagrt had a bug where the logging level was set to "Info" as soon as "–verbose" was passed, but the value of "–verbose" was not even checked.

9.3 0.6.4

Bugfix release for fixing:

- libimagrt produced the editor command without taking arguments into account.
- imag−init creates ~/.imag but not ~/.imag/store.
- Fix editor setup in libimagrt to use /dev/tty as stdin for editor, so terminal-editors do not trash the terminal

9.4 0.6.3

Bugfix release for fixing:

- libimagstore got another fix with the file parsing, as the std :: str :: Lines iterator takes empty lines as no lines.

9.5 0.6.2

Bugfix release for fixing:

- imag−diary did not recognize the "-d DIARY" setting.
- A parsing error in libimagstore, which caused the parsing of entries with a line "—" in the content part to fail, was fixed.
- The bugfix above introduced another bug which caused entries to be rewritten in one line when accessing them. This was fixed.
- imag−diary did not properly set "minute" and "second" when creating "hourly" or "minutely" entries.
- Version numbers for all crates as well as in the docs were updated to "0.6.2".

9.6 0.6.1

Bugfix release for fixing two severe bugs in imag−init:

- imag−init created the git directory inside the imag directory. Fixed by defaulting to {imag directory}/.git.
- imag−init was buggy as it did not include the imagrc.toml file in the release, thus building it from crates.io failed

9.7 0.6.0

- Major changes
  - The config infrastructure of libimagstore was removed, as it was unused.
  - The iterators of libimagstore were improved and are now abstract over all iterator types. For example, all iterators over StoreId can now be transformed into a StoreGetIterator.
  - imag−log was introduced
  - imag−init was introduced
  - libimagdiary supports second-granularity now.
  - libimagstore :: store :: Store :: retrieve_copy was renamed to libimagstore :: store :: Store :: get_copy, which describes the semantics of the function way better.
  - libimagentryutil was introduced, a library for helpers for libimagstore :: store :: Entry handling and writing extension-writing.
  - imag−edit was introduced
  - imag−diary got second-granularity support in the CLI.
- Minor changes
  - Internals were refactored from matching all the things into function chaining
  - The toml−query dependency was updated to 0.5.0
  - imag−timetrack list lists with a table now
  - imag−timetrack stop now stops all running tags if none are specified
  - The toml−query dependency was updated to 0.6.0
  - ResultExt::map_err_trace_exit () was removed in favour of ResultExt::map_err_trace_exit_unwrap().
  - imag−view shows content by default now. Use −C to hide the content.
  - kairos dependency was updated to 0.1.0
- Bugfixes
  - libimagbookmark contained a type which wrapped a FileLockEntry from libimagstore. This was considered a bug and was fixed.
  - We depended on a crate which was licensed as GPLv2, which would yield imag GPL as well. The dependency was removed.
  - The imag crate prints the “command filed” error message to stderr now. It also prefixes the subcommand with imag−<command> for better user experience.
  - libimagnotes did not set the note name in the header of the note.
  - imag−mv automatically fixes links when moving an entry in the store.

  - imag−log listed non-log entries (normal diary entries) before, was changed to only list log entries.

9.8 0.5.0

- Major changes
  - imag−counter and libimagcounter was removed.
  - imag−mv was introduced
  - imag−view uses positional args now
  - imag−view uses the configuration file now to find the command to call for viewing the entry. This way one can view the entry in an editor or the browser or on the toaster.
  - The logger is now able to handle multiple destinations (file and "-" for stderr)
  - imag−store can dump all storeids now
  - imag−annotate was introduced
  - imag−diagnostics was added
  - The runtime does not read the config file for editor settings anymore. Specifying an editor either via CLI or via the $EDITOR environment variable still possible.
  - imag−contact was added (with basic contact support so far).
  - imag−habit was introduced
  - imag−link commandline was redesigned to be easier but with the same features.
- Minor changes
  - libimagentryannotation got a rewrite, is not based on libimagnotes anymore. This is minor because libimagentryanntation is not yet used by any other crate.
  - imag now reads the IMAG_RTP environment variable before trying to access $HOME/.imag for its runtimepath.
  - libimagnotification was introduced, though not yet integrated into the CLI tools
- Bugfixes
  - Store :: entries () does not yield StoreIds which point to directories anymore, only StoreIds pointing to files.
- Stats
  - 227 commits
  - 51 merge-commits / 176 non-merge commits
  - 2 contributors
  - 186 files changed
  - 6707 insertions(+) / 3255 deletions(-)

9.9 0.4.0

- Major changes

  - The libimagstore :: toml_ext module was removed. The toml_query crate should be used as a replacement. Its interface only differs in few places from the old libimagstore :: toml_ext interface.
  - The codebase was moved to a more tree-ish approach, where several subdirectories were introduced for different types of crates
  - The documentation got a major overhaul and was partly rewritten
  - The logger is now configurable via the config file.
  - The error handling of the whole codebase is based on the error_chain now. libimagerror only contains convenience functionality, no error-generating macros or such things anymore.
  - imag−diary can now use a configuration in the imagrc.toml file where for each diary there is a config whether entries should be created minutely or hourly (or daily, which is when specifying nothing).

- New

  - libimagentrygps was introduced
  - imag−gps was introduced
  - imag−grep was introduced
  - The imag command now passes all arguments properly to the called subcommand

- Fixed bugs

  - The config loading in libimagrt was fixed.
  - libimagentrylink used imag as the location for putting links in entries. This is not allowed because this namespace is reserved for the store itself. This bug was fixed, links are now located in the links namespace in the header of an entry.
  - Store :: delete () did only check the store-internal cache whether an entry exists, but not the filesystem. This was fixed.

- Minor changes

  - If building from a nix−shell, the mozilla rust overlay is expected to be present
  - Unused imports in the codebase were removed
  - Compiler Warnings were fixed
  - We specify inter-dependencies via path and variable now, so one can build the 0.3.0 release from the checkout of the codebase.
  - The imag binary was refactored and rewritten, the crossbeam dependency was removed.
  - The Makefile was removed as cargo is powerful enough to fit our needs
  - libimagstore :: storeid :: StoreId :: is_in_collection () was added
  - The libimagentrylink is now rudimentarily tested
  - We compile with rustc 1.17, 1.18, .., nightly
  - The imag−store binary now uses positional arguments in its CLI
  - The “toml-query” dependency was updated to 0.3.1
  - imag−timetrack track is now able to parse “now”, date-only start/stop dates and date-time start/stop times.
  - libimagnotes does not longer wrap store types but extend them.
  - imag−notes uses positional arguments now.

  - libimagentrylist does not export a CLI helper module anymore.

- Stats
  - ~325 commits
  - 82 merge-commits / 243 non-merge commits
  - 2 contributors
  - 447 files changed
  - 9749 insertions(+) / 7806 deletions(-) (Surely because of the reorganization of the entire codebase)

9.10 0.3.0

Note: As this file was written *after* the 0.3.0 release, we simply list the merges here instead of explaining what changed.

- Merges
  - d14c972 matthiasbeyer/release-commits-import
  - f6a1c7d matthiasbeyer/make-check
  - 0404b24 matthiasbeyer/update-deps
  - 85e95d1 matthiasbeyer/readme-rewrite
  - fa64c2d matthiasbeyer/libimagstore/store-id-cmp-without-base
  - 0a04081 matthiasbeyer/cargo-rustc-codegen-units
  - a4db420 matthiasbeyer/cargo-workspaces
  - 8bacdb4 matthiasbeyer/libimagref/remove-unused
  - 13c57aa matthiasbeyer/imag-link/reduce-unwraps
  - e70fdc6 matthiasbeyer/libimagentrytag/remove-impl-tagable-on-fle
  - 1db063f Stebalien/master
  - a6a7e43 mario-kr/add_shell-completion
  - 002c50a matthiasbeyer/clap-completion
  - b210b0e matthiasbeyer/libimagstore/entry-eq
  - fe1c577 matthiasbeyer/libimagstore/extract-toml-functionality
  - 4ca560a matthiasbeyer/travis-use-old-rustc
  - 0310c21 rnestler/libimagdiary/refactor_from_store_id
  - 9714028 matthiasbeyer/clap-recommend-versions
  - 2003efd matthiasbeyer/imag-mail/init
  - 7c7aad9 matthiasbeyer/libimagentrylink/fix-docu-typo
  - 0dd8498 matthiasbeyer/update-deps
  - 9375c71 matthiasbeyer/makefile-check-outdated
  - 23a80ee matthiasbeyer/imag-link/external-link-remove-arg
  - 4a821d7 matthiasbeyer/rust-beta-remove-top-level-cargotoml
  - 0cf5640 mario-kr/add_workspace-support
  - c96e129 matthiasbeyer/libimagrt/logger-pub
  - 2c4946a matthiasbeyer/remove-for-focus-shift
  - 9d7a26b matthiasbeyer/libimagrt/dbg-fileline-opt
  - 6dbecbd matthiasbeyer/libimagrt/config-types-pub

- 03a90c9 matthiasbeyer/cleanup-bash-compl-gen
- 6f564b5 matthiasbeyer/love-to-defaultnix
- ce36b38 matthiasbeyer/fix-imag-bin-build
- cd684b0 matthiasbeyer/travis-opt
- 636bfbb matthiasbeyer/imag-link/list-internal-only
- 1e3193e matthiasbeyer/imag-ruby
- 71e1a4c matthiasbeyer/libimagerror/fix-warnings
- b03d1b5 matthiasbeyer/libimagstore/fix-warnings
- 0a417aa matthiasbeyer/libimagruby/fix-warnings
- 55ea7f8 matthiasbeyer/readme-updates
- ddc49de matthiasbeyer/libimagruby/fix-macro
- df0fa43 matthiasbeyer/imag-tag/remove-warning
- 3c7edcf matthiasbeyer/update-regex
- 15b3567 matthiasbeyer/workspace-fix-missing-doc
- 6585677 matthiasbeyer/libimagentryfilter/remove-unused-import
- 2ca89b7 matthiasbeyer/workspace-fix
- 63ffbf6 matthiasbeyer/libimagstore/eliminate-header-type
- 3ffedec matthiasbeyer/remove-warnings
- 4d1282d matthiasbeyer/libimagruby/impl-retrieve-for-mod
- c43538d matthiasbeyer/ruby-build-setup
- 2beb795 matthiasbeyer/revert-871-ruby-build-setup
- b67b6f5 matthiasbeyer/libimagstore/doc
- dc1c473 matthiasbeyer/libimag-todos
- bb126d5 matthiasbeyer/libimagruby/api-brush
- b50334c matthiasbeyer/libimagrt/doc
- 54655b9 matthiasbeyer/libimaginteraction/unpub-fn
- e33e5d2 matthiasbeyer/libimagannotation/init
- f3af9e0 matthiasbeyer/clap-bump
- ef07c2c matthiasbeyer/libimagstore/verify-panic
- a0f581b matthiasbeyer/libimagentryedit/dont-impl-for-fle
- 84bcdc6 matthiasbeyer/libimagnote/note-doesnt-need-to-be-tagable
- 85cb954 matthiasbeyer/less-fold-more-defresult
- c4bd98a mario-kr/makefile_use_workspaces
- 3a0166b matthiasbeyer/libimagruby/error-types
- 5d4ef8e matthiasbeyer/libimagstore/non-consuming-update
- e615ec0 matthiasbeyer/add-libruby-travis-dep
- 63faf06 matthiasbeyer/fix-warnings
- 6f6368e matthiasbeyer/travis-fixes
- 9396acc matthiasbeyer/superceed-898
- 6fa281a matthiasbeyer/redo-ruby-build-setup
- 5b93f38 matthiasbeyer/libimagstore/storeid-exists-interface-result
- 03f17b8 matthiasbeyer/libimagentrylink/annotations
- 25a3518 matthiasbeyer/libimagentrylink/fix-exists
- 7e3c946 matthiasbeyer/libimagutil/fix
- 8eaead5 matthiasbeyer/fix-build-quick

- 241f975 matthiasbeyer/libimagentryedit/remove-unused-imports
- c74c26c matthiasbeyer/fix-readme-links
- 878162f matthiasbeyer/libimagstore/store-id-tests
- 1da56c6 matthiasbeyer/prepare-0.3.0
- 4257ec1 matthiasbeyer/update-toml
- a5857fa matthiasbeyer/libimagstore/configuration-tests
- 4ba1943 matthiasbeyer/add-dep-ismatch
- 5ba2568 asuivelentine/master
- dd24ce8 matthiasbeyer/revert-854
- bb9ff5b matthiasbeyer/remove-hooks
- 08f43c8 matthiasbeyer/update-toml-query
- 16a12af matthiasbeyer/libimagentrydate/init
- 1b15d45 matthiasbeyer/libimagentrydate/fix-header-location
- 4fff92e matthiasbeyer/libimagmail/use-email-crate
- ef82b2a matthiasbeyer/add-missing-license-header
- 15b77ac matthiasbeyer/libimagentrytag/clap-validators
- a9d2d7c matthiasbeyer/libimagstore/fs-memory-backend-as-dependency-injection
- c4d4fe9 matthiasbeyer/libimagstore/remove-todo-comment
- 71e3d3d matthiasbeyer/libimagentrytag/validator-helper-enhancement
- bc95c56 matthiasbeyer/libimagstore/fs-abstraction-pub
- f487550 matthiasbeyer/libimagstore/storeid-local-part-altering
- cd99873 matthiasbeyer/libimagstore/io-backend
- e75c37f matthiasbeyer/libimagstore/io-backend-knows-format
- d33b435 matthiasbeyer/libimagstore/all-entries
- f8ed679 matthiasbeyer/libimagstore/backend-replacement
- 2c97d6f matthiasbeyer/libimagstore/embellishments
- 17bab5b matthiasbeyer/libimagstore/fixes
- 2b77064 matthiasbeyer/libimagrt/fixes
- c9d03fc matthiasbeyer/update-travis
- 22a4dc0 matthiasbeyer/libimagrt/cleanup
- b47972b matthiasbeyer/imag-store-dump
- 1b88c22 matthiasbeyer/libimagentrycategory/init
- 7dea53c matthiasbeyer/libimagannotation/add-doc
- c71b707 matthiasbeyer/libimagannotation/add-is_annotation
- b3e7f09 matthiasbeyer/libimagtimetrack
- c75cfe4 matthiasbeyer/imag-link/fix-panic
- e80608c matthiasbeyer/libimagstore/fix-file-length-setting
- b4d0398 matthiasbeyer/imag-link/export-consistency-check
- f041fb3 matthiasbeyer/fix-dep-rustc-version
- 297eeb1 matthiasbeyer/remove-nix-deps
- bee4e06 irobert91/imag-link/rewrite-tests
- afc5d1f matthiasbeyer/update-toml-query
- 58047d3 matthiasbeyer/libimagtimetrack-to-libimagentrytimetrack
- 3c07f47 matthiasbeyer/libimagtimetrack/more-features
- 3767d8d matthiasbeyer/update-chrono

  - c9360a4 matthiasbeyer/imag-link/test-utils-to-libimagutil
  - e4f8d4e matthiasbeyer/imag-tag/tests
  - fc5bbc3 matthiasbeyer/libimagstore/glob-iterator-fix
  - ec1c1e8 matthiasbeyer/bin-refactor
  - 4b07c21 matthiasbeyer/todo
  - 057d919 matthiasbeyer/imag-timetrack
  - 0f436d5 matthiasbeyer/doc-overhaul
  - a1289cc matthiasbeyer/update-readme

- Stats:

  - 127 merged branches
  - 8 contributors

9.11 0.2.0

- Complete rewrite of 0.1.0 with new architecture and "modules" instead of monolithic approach. Can be considered first version.

9.12 0.1.0

- Initial version, nothing special.