xettel

Release 0.3.1

xaltsc

Jan 09, 2022

CONTENTS:

`xettel` is a Zettelkasten management software backed by xapian.

`xettel` provides a way to store your Zettels in a Zettelkasten that allows for full-text search using `xapian` as its backend. The interface `xettel` provides searching is similar to that of notmuch. One of the core design principles of `xettel` is to be completely agnostic as to the format the end-user stores their Zettels in. Should they wish to store their Zettels in an exotic format, say, `.docx`, if they are able to provide a class that extracts the information for `xettel`, they will be able to use the software just like with simple `markdown` files.

Beyond providing a search interface, `xettel` also provides ways to export the Zettels in another format, for instance, `.html`, health checks such as weak-connectivity of the whole Zettelkasten and can also produce a map of the links.

CHAPTER

ONE

INSTALLATION

1.1 Requirements

`xettel` needs Python >=3.7 and the `xapian` librairy and Python bindings. For instance, on void-linux, run the following command:

Todo: Check this is the right version. May not run because of type-hints. If it fails, try python 3.9.

```
$ sudo xbps-install libxapian xapian-core-*
```

Once this is done, clone the repository and install it with pip

```
$ hg clone https://hg.sr.ht/~xaltsc/xettel
$ cd xettel
$ pip install .
```

You can check that `xettel` has been installed by running

```
$ xettel --help
```

CHAPTER

TWO

QUICKSTART

2.1 How `xettel` works

`xettel` is a Zettelkasten management software. It uses the folder/file metaphor to represents Zettelkästen and Zettels, that is, your Zettelkasten will be a folder, and your Zettels will be the files in this folder. Beyond that, `xettel` tries to be as agnostic as possible concerning your Zettels. Even though there is a default implementation for Zettels in Markdown, you may store your Zettels in any format you like, provided you can supply a way to extract the data. See *Extending xettel* for more information.

However, in order to perform some graph-related checks on your Zettelkasten and in order to find your files, filenames must abide by a few rules:

1. Filenames of Zettels must be of the form `UID-name.ext` where `UID` is either a 12-digit long number in base 10 or a 8-digit long number in base 36 (case doesn't matter), `ext` must be the extension defined in your class, i.e., for Markdown files using the default implementation, it must be `md`, and `name` has no special constraints. By default, `xettels` creates Zettels with an uppercase number in base 36 for the UIDs based on the time they were generated. In base 10, this uid is just `YYMMDDHHmmSS`.
2. In order to perform graph checks and to plot the graph of the Zettelkasten, `xettel` assumes that your index file (i.e. the main entry point of your Zettelkasten) has the UID `0` (thus, in base 36, a filename for the index file may be `00000000-index.md`).

> **Warning:** Pay attention to the fact that UIDs are stored solely in the filename. You may not have two Zettels whose filenames begin with the same number, even if one is in base 36 and the other in base 10.

In order to keep the graph tidy (as much as it is possible) when plotting it, there is an extra rule concerning this time the content of your Zettels. If your implementation has tags (as has the default markdown implementation), you may have some Zettels that work has entry points to some subjects. For instance, you may have a Zettel `uid-norway.md` that has links to Zettels pertaining to Norway. Such Zettels are called hubs and should be tagged with `hub`. This will set them a bit further from the other zettels when plotting, especially when the link is between two hubs.

As said above, `xettel` is quite agnostic as to how you store your Zettels, however, when using the default Markdown implementation or one of its subclasses, please refer to the section *Using the default Markdown implementation* (cf. infra) for usage.

Because viewing your notes in the source format is not always pleasant, `xettel` allows you to export them in a given format. However, this is implementation dependant, and therefore is explained in *Using the default Markdown implementation* for the default implementation and in *Extending xettel* in full generality.

2.2 Setting up `xettel`

As explained above, in order to use `xettel`, you must first chose a folder to store your Zettels in, an implementation, and you must also define a template to create new Zettels with `xettel new`.

The config folder is by default at `~/.config/xettel`. All of the config data is expected to be in the TOML file `config.toml` in this folder. For instance, here's a default `config.toml` file.:

```
[Zettelkasten]
directory="/path/to/zettelkasten/folder"
template="/path/to/template/folder/template.ext"
zk_impl = "markdown"
```

This config file may contain additional fields for use with the implementations.

Todo: Set up a way to pick the config folder respecting XDG stuff.

Don't forget to create a Zettel with uid `00000000` and you should be able to use `xettel`. Further usage is described in *Command-line usage*.

2.3 Using the default Markdown implementation

As a concrete example of the above, let us show how to use the default Markdown implementation defined at *xettel.impl.markdown.ZettelMD.ZettelMD*.

Files that are meant to be used with this class are standard Markdown files with YAML headers. Their expected extension is `.md`. For instance, here is a template:

```
---
title: Title
tags: [tag1, tag2]
abstract: abstract
---

CONTENT
```

Warning: The YAML header must be **valid**, empty fields, for instance, cause bugs.

YAML fields may be anything, the current implementation only parses the fields `title`, `tags` and `abstract`, but future implementations may use additional fields as well as classes derived from it.

Links in this format must verify the regex `(?<=\[#)([0-9]{12}|[0-9a-zA-Z]{8})(?=\])`, i.e. they must be of the form `[#UID]` where UID is either an 8-digit long number in base 36 or a 12-digit long number in base 10.

2.3.1 Exporting

This class levies `pandoc` to export Zettels to HTML. In order to do so, a new folder in created within the Zettelkasten folder called `export`. `xettel` calls `pandoc` with a number of metadata set for use with a template, however, it does not tell pandoc where to find templates or anything, making it unusable. To make it usable, you should create you own class, see *Extending xettel*.

CHAPTER

THREE

COMMAND-LINE USAGE

3.1 xettel

Main function for the cli interface.

This function sets the config, loads implementations and sets the variables `xettel.config.Z_CLASS` and `xettel.config.ZK_CLASS` according to the `zk_impl` field of the config file.

Args: config (str): the path to the config file. ~ is expanded. dir (str): the path to the Zettelkasten folder.

```
xettel [OPTIONS] COMMAND [ARGS]...
```

Options

-c, **--config** <config>
: config file

-d, **--dir** <dir>
: Zettelkasten directory

3.1.1 count

count zettels matching query

```
xettel count [OPTIONS] QUERY...
```

Arguments

QUERY
: Required argument(s)

3.1.2 edit

Edit zettel matching query. The query must match exactly one zettel

```
xettel edit [OPTIONS] IDENTIFIER...
```

Options

-e, --editor <editor>
editor to write in

Arguments

IDENTIFIER
Required argument(s)

Environment variables

EDITOR

Provide a default for *-e*

3.1.3 export

export the Zettelkasten to html

```
xettel export [OPTIONS]
```

Options

-f, --force
force rewriting all exports

3.1.4 genmap

generate a map of the zettelkasten

```
xettel genmap [OPTIONS]
```

3.1.5 gentags

generate tag pages

```
xettel gentags [OPTIONS]
```

Options

-d, --delete
delete tag pages that match no zettel

3.1.6 health

check the health of the zettelkasten

```
xettel health [OPTIONS]
```

3.1.7 new

Create a new zettel

```
xettel new [OPTIONS] NAME
```

Options

-e, --editor <editor>
editor to write in

--noeditor
do not use an editor

--register
register newly created file

-u, --returnuid
return B36 newly created uid (implies –no-editor anf –register)

-t, --template <template>
template for the new file

Arguments

NAME
Required argument

Environment variables

EDITOR

Provide a default for *-e*

3.1.8 search

search in the database

```
xettel search [OPTIONS] QUERY...
```

Options

-o, --output <output>
output options

Options normal | uid | tags | filename | backlinks | map | json

-s, --sort
sort by most recent

-n, --number <number>
Number of items to query for

Arguments

QUERY
Required argument(s)

3.1.9 updatedb

update the db from the zettelkasten folder

```
xettel updatedb [OPTIONS]
```

Options

-b, --fetchbacklinks
when used together with the option –one, resolvebacklinks as well

-1, --one <one>
only update a single file

-f, --force
force overwriting database

-x, --delete
purge database of files not present anymore

3.2 Subcommands

3.2.1 Interfacing with Xapian

xettel updatedb

update the db from the zettelkasten folder

```
xettel updatedb [OPTIONS]
```

Options

-b, --fetchbacklinks
when used together with the option –one, resolvebacklinks as well

-1, --one <one>
only update a single file

-f, --force
force overwriting database

-x, --delete
purge database of files not present anymore

xettel search

search in the database

```
xettel search [OPTIONS] QUERY...
```

Options

-o, --output <output>
output options

Options normal | uid | tags | filename | backlinks | map | json

-s, --sort
sort by most recent

-n, --number <number>
Number of items to query for

Arguments

QUERY
Required argument(s)

In this command, the query is either `*`, meaning the whole database, or a valid `xapian` query. Much like `notmuch`, a query is a string made of prefixed terms or free terms joined together with logical operators `OR`, `AND` and so on. Please refer to Xapian documentation for the explicit format of the queries.

Following `xettel`'s agnostic philosophy, prefixes may be anything, as they are set in the implementations. However, there are a few conventions and some of the prefixes are set in the base class so are always available.

uid This prefix refers to the UID of a document, its value being a number in base 10 or 36. There is no need to pad it so that it fits in 8 or 12 characters contrarily to filenames. Note however that since searching is done on strings, you may either use a properly formatted UID or a number without leading zeroes. This is a boolean prefix.

text Searches in the text of Zettels. The prefix is optional.

title Searches in the title of Zettels.

abstract Searches in the abstract of Zettels.

tag Searches among tags. This is a boolean prefix.

filepath Searches among the file path of the Zettels.

filename Searches among the file names of the Zettels.

links Searches the Zettels that link to the provided uid. Conventions follow the same rules as for the prefix `uid`. This is a boolean prefix.

backlinks Searches the Zettels that are linked from the provided uid. Conventions follow the same rules as for the prefix `uid`. This is a booolean prefix.

xettel count

count zettels matching query

```
xettel count [OPTIONS] QUERY...
```

Arguments

QUERY
Required argument(s)

Queries are the same as for `xettel search`.

3.2.2 Editing Zettels

xettel new

Create a new zettel

```
xettel new [OPTIONS] NAME
```

Options

-e, --editor <editor>
editor to write in

--noeditor
do not use an editor

--register
register newly created file

-u, --returnuid
return B36 newly created uid (implies –no-editor anf –register)

-t, --template <template>
template for the new file

Arguments

NAME
Required argument

Environment variables

EDITOR

Provide a default for *-e*

This command generates the new ID for the Zettel with the current time, as `YYMMDDHHmmSS` and copies the template file to `UID-NAME.ext`, with the UID in uppercase base 36.

xettel edit

Edit zettel matching query. The query must match exactly one zettel

```
xettel edit [OPTIONS] IDENTIFIER...
```

Options

-e, --editor <editor>
editor to write in

Arguments

IDENTIFIER
Required argument(s)

Environment variables

EDITOR

Provide a default for *-e*

IDENTIFIER is a query that must match a single Zettel.

3.2.3 Exporting

xettel export

export the Zettelkasten to html

```
xettel export [OPTIONS]
```

Options

-f, --force
force rewriting all exports

This command calls the `export` method of implementations. Unless `-f` is specified, it does not export Zettels if the destination file is newer than the source Zettel.

xettel gentags

generate tag pages

```
xettel gentags [OPTIONS]
```

Options

-d, --delete
delete tag pages that match no zettel

This method calls the `tag_export` class method of implementations.

3.2.4 Graph aspects of a Zettelkasten

xettel health

check the health of the zettelkasten

```
xettel health [OPTIONS]
```

This command checks if the Zettelkasten is weakly-connected to the root, i.e. the Zettel with uid `0`.

xettel genmap

generate a map of the zettelkasten

```
xettel genmap [OPTIONS]
```

This command generates an svg clickable map of the Zettelkasten in the export folder. It uses `graphviz` in order to do so. Some special tweaking is done for edges between two hubs in order to render the produced map more legible.

If you want for this map to be included in your Zettelkasten, create a new Zettel showing it, for instance:

```
---
title: Map of the zettelkasten
tags: [meta]
---

<base target='_blank'>
<iframe id="map" src="map.svg">
web browser not supporting svg
</iframe>
```

CHAPTER

FOUR

CONFIGURATION

CHAPTER

FIVE

EXTENDING XETTEL

xettel is built in such a way that it is agnostic as to the way you store your Zettels. By default, it only provides a single class, *xettel.impl.markdown.ZettelMD.ZettelMD* that provides a very elementary way of handling Markdown Zettels. End users should, and are encouraged to, define their own classes fitting their own needs.

In the extra/config/impl/ folder of the source code, there is the class that I use, which, for instance, provides support for LaTeX snippets and even tikz-cd snippets.

5.1 How to make xettel aware of my implementation

Before entering into the details of how to code your own implementation, let us detail how xettel will be aware of your new code.

First, all your extra code should reside in the impl subfolder of your configuration folder (usually ~/.config/xettel). At runtime, xettel imports all the files within this folder.

xettel keeps a registry of all available implementations in *xettel.implRegister*.

In order to register your class, you should decorate it with *xettel.implRegister.register_Z()*. The name you give it as a parameter is the name you should give in your configuration to the field Zettelkasten.zk_impl in order to use it.

Should you wish to not use the folder/file metaphor, you will also have to register a Zettelkasten class using *xettel.implRegister.register_ZK()*.

5.2 How to code a new implementation

We only describe how to make your own implementation using the folder/file metaphor. Other use-cases are exotic and require a more in-depth understanding of xettel that you can achieve by reading the source code.

Basically, your new implementation should be a subclass of *xettel.base.ZettelFile.ZettelFile*. This class only reads the file and some of its attributes. From there, there are several ways you can go.

Note: If you just need to work with Markdown Zettels, you may as well make your class a subclass of *xettel.impl.markdown.ZettelMD.ZettelMD*. Even if you don't want to use Markdown, this class is a good example to see what to implement and how to do it.

In order to make things more portable, at runtime, the class has access to the config file which is stored as a dictionnary in *xettel.config.config*.

5.2.1 Parsing a file

If you want to have your Zettels stored in a particular format, the two methods you should overload are *xettel.base.ZettelFile.ZettelFile.parse_attributes()* and *xettel.base.ZAbstract.ZettelAbstract.parse_links()*.

When overloading *parse_attributes()*, the raw text is available at the `'bare_contents'` key of the *xettel.base.ZAbstract.ZettelAbstract.attributes* dictionnary.

You may store the attributes in this dictionnary under any name you want, however, there are some conventions that are used when entering the data in `xapian`.

First of all, the entries at `'uid'`, `'backlinks'` and `'links'` are reserved and should not be used. The contents of the Zettel (i.e. everything that is not metadata) should be stored under the key `'text'`, the title under `'title'`, tags under `'tags'` as a list of strings, the abstract under `'abstract'`.

When overloading this method, you should **NOT** parse links.

Next, when overloading *parse_links()*, you may safely ignore the `links` parameter or do nothing with it. In the *ZettelMD* class, it's only there for typechecking purposes. This method should find links within the text, convert them to *int* and pass them to the *xettel.base.ZAbstract.ZettelAbstract.parse_links()* method.

5.2.2 Exporting

Probably the major reason for having built `xettel` modularly is to allow end users to have a large control over the export process.

The three methods involved in this process are *xettel.base.ZettelFile.ZettelFile.export()*, `xettel.base.ZettelFile.issue_export_cmd()` and *xettel.base.ZettelFile.ZettelFile.tag_export()*, the latter being there to generate tag pages. All these methods are static methods.

The role of the *export()* method is to prepare the metadata to be passed to `issue_export_cmd()` and, check if the destination file is newer than the source and call `issue_export_cmd()` otherwise, or if `force` is true. The actual command is executed from `issue_export_cmd()`.

There are two kinds of metadata that is passed to `issue_export_cmd()`: `'env'` is a dictionnary of strings that consists of the environement variables; `'zinfo'` is a dictionnary of strings that consists of parameters to pass to the command.

When using, for instance, `pandoc`, you may specify filters, templates, etc., in the `issue_export_cmd()` method.

When overloading *tag_export()*, you have to generate a complete document before sending it to `issue_export_cmd()`.

5.2.3 Adding new search prefixes

Modifying the search process involves dealing directly with `xapian`. Please read the documentation and the documentation on Python bidings to understand how all of this works.

`xapian` stores data in two different ways: the document and the terms associated to it. The document consists of arbitrary data and is not a priori searchable, its main purpose is to allow for retreiving data after having found it. The terms are what `xapian` searches.

Saving data and setting search terms is done by the *xettel.base.ZettelFile.ZettelFile.to_xapian()* method.

Setting the document

`xettel` stores the data of a Zettel as a JSON dictionnary containing the attributes. This is done in the method *xettel.base.ZettelFile.ZettelFile.prepare_data_for_xapian()*, which adds the data of the links, backlinks and UID and prunes the bare content.

Setting the search terms

The method *to_xapian()* individually sets some universal search terms, such as links, uids, backlinks. For less universal terms, such as titles, abstract, etc., it calls the method *xettel.base.ZettelFile.ZettelFile.index_attributes()*.

This methods iterates over the keys of *attributes* and works in conjuction with *get_indexing_function()* to get a function that sets the terms using the provided `TermGenerator`. Note that throughout the whole process of creating terms for the Zettelkasten, a single term generator is used.

Be careful, when creating new prefixes, that it is expected that they follow some conventions, as explicited in xapian documentation.

CHAPTER

SIX

XETTEL API

6.1 Abstract and Base Classes

6.1.1 ZAbstract

exception xettel.base.ZAbstract.**UnresolvedLinks**(*unresolved: list[typing.Tuple[int, int]]*)
Exception for unresolved links

Parameters **unresolved** (*list[UnresolvedListItem]*) – a list of tuples of ints (source of the link, destination of the link).

unresolved
a list of tuples of ints (source of the link, destination of the link).

Type list[UnresolvedListItem]

class xettel.base.ZAbstract.**ZettelAbstract**(*parent:* xettel.base.ZAbstract.ZettelkastenAbstract, *uid: int*)
Abstraction for a single Zettel

Parameters

- **parent** (ZettelkastenAbstract) – Zettelkasten in which the Zettel is stored.
- **uid** (*int*) – uid of the Zettel

parent
the Zettelkasten in which this Zettel is stored.

Type *ZettelkastenAbstract*

__uid
the uid of this Zettel.

Type int

attributes
dict[str, Any]: the attributes of this Zettel

__links
set of the uids of the Zettels this Zettel points to.

Type set[int]

__backlinks
set of the uids of the Zettels that point to this Zettel.

Type set[int]

Raises TypeError – if the parent is not an instance of ZettelkastenAbstract or if the uid is not an int.

add_backlink(*uid: int*) → None
Add an uid to the set of backlinks.

Parameters **uid** (*int*) – an uid of a Zettel.

Raises **TypeError** – if uid is not an int.

fetch_backlinks() → None
Fetch backlinks for current zettel.

This method is more efficient for a single zettel.

get_backlinks_uids() → set[int]
Get uids of Zettels that points to this Zettel.

Returns a set of the uids of the Zettels that points to this one.

Return type set[int]

get_links_uids() → set[int]
Get uids of the links in this zettel.

Returns a set of the links from this zettel

Return type set[int]

get_uid() → int
Get uid of Zettel.

Returns the uid of the Zettel

Return type int

get_uid_36() → str
Get uid of the Zettel in base 36.

Returns the uid of the Zettel, in base 36.

Return type str

initialise_contents() → None
Orderly initialise contents.

This method first parses attributes then parses the links.

abstract parse_attributes() → None
Parse attributes

abstract parse_links(*links: collections.abc.Collection[int] = []*) → None
Parse links with the given data

The abstract method defined here just sets the private attribute `__links` to the ones given by the argument `links`.

Parameters **links** (*Collection[int]*) – a collection of ints.

propagate_backlinks() → None
Tries to set backlinks in zettels linked from this zettel.

Raises ***UnresolvedLinks*** – if some links are unresolved, i.e. they point nowhere.

class xettel.base.ZAbstract.**ZettelkastenAbstract**
Abstraction for a Zettelkasten

zettels
a dictionnary of Zettels where each zettel is supposed to be indexed by its uid.

Type dict[int, ZettelType]

add_zettel(*zettel: xettel.base.ZAbstract.Z*) → None
Add Zettel within the Zettelkasten, eventually replacing already existing Zettel with same UID

Parameters **zettel** (*ZettelType*) – a Zettel

initialise_contents(*setbacklinks: bool = True*) → None
A posteriori setting of the contents.

Optionally set backlinks as well when useful. Defaults to True.

Parameters **setbacklinks** (*bool*) – whether to set backlinks or not

set_backlinks() → None
Tries to set all backlinks, raise an exception if some links cannot be resolved.

This method is more efficient than using fetch_backlinks on all zettels

xettel.base.ZAbstract.**intkey**(*key: Union[int, str]*) → int
Convert a valid key to an int key

Parameters **key** (*ZKey*) – A valid key, either a alphanumeric string which represents a number in base 36 or an int.

Returns An int key.

Raises **TypeError** – If the provided key is not valid.

6.1.2 ZettelFile

class xettel.base.ZettelFile.**ZettelFile**(*parent: Zettelkasten*, *filename: str*)
Abstraction for a single Zettel file

Parameters

- **parent** (Zettelkasten) – the Zettelkasten in which this Zettel is stored. The Zettelkasten must be a subclass of ZettelkastenFolder.
- **filename** (*str*) – the bare filename (not the whole path!). The filename must be of the form `$UID-*` where $UID is either a base36 8-char long string or a base10 12-digit long string.

extension
the extension of the files. This is a class attribute.

Type str

abstract static export(*zettel:* xettel.base.ZAbstract.ZettelAbstract, *force: bool = False*) → None
Exports Zettels.

The abstract method defined here does nothing.

Parameters

- **zettel** (*ZA.ZettelAbstract*) – a Zettel
- **force** (*bool*) – whether to override the destination even if it is newer than the Zettel's file.

get_indexing_function(*key: str*) → Callable[[xapian.Document, xapian.TermGenerator, Any], None]
Returns the function needed to index data from the attributes given a lowered and whitespace-stripped (as in str.strip()) key.

Functions returned by this method must have the following signature: `(xapian.Document, xapian.TermGenerator, value) -> None`.

Parameters **key** (*str*) – the key to the indexing function

Returns The indexing function corresponding to the key.

index_attributes(*document: xapian.Document*, *indexer: xapian.TermGenerator*) → None
Indexes attributes of the Zettel in the Xapian database

Parameters

- **document** (*xapian.Document*) – the document in which to index attributes.
- **indexer** (*xapian.TermGenerator*) – the term generator with which xettel will generate terms for the document.

static index_path(*doc: xapian.Document*, *idxr: xapian.TermGenerator*, *path: str*) → None
Function that generates terms for a xapian document with its path.

Parameters

- **doc** (*xapian.Document*) – a Xapian document
- **idxr** (*xapian.TermGenerator*) – a Xapian term generator.
- **path** (*str*) – a path

abstract static issue_export_cmd(*source: str*, *dest: str*, *env: dict[str, str]*, *zinfo: dict[str, str]*) → None
Execute the command to export Zettels.

The abstract method defined here does nothing.

Parameters

- **source** (*str*) – path of the Zettel
- **dest** (*str*) – where to export the Zettel
- **env** (*dict[str, str]*) – dictionnary of environement variables to pass to the command.
- **zinfo** (*dict[str,str]*) – additional information to feed the command, probably directly among the arguments of the command.

abstract parse_attributes() → None
This method assumes the file exists

prepare_data_for_xapian() → dict[str, typing.Any]
Builds a data dict to store in a xapian.Document

Returns A dictionnary of keys and attributes to feed to Xapian.

classmethod set_queryparser_prefixes(*queryparser: xapian.QueryParser*) → None
Sets the prefixes for searching the metadata defined by this class into the queryparser.

The method defined here does nothing.

Parameters **queryparser** (*xapian.QueryParser*) – the queryparser to feed information to.

abstract classmethod tag_export(*tag: str*, *filecontents: list[typing.Tuple[str, str]]*) → None
Exports a page of a list of Zettels having this tag.

The abstract method defined here does nothing.

Parameters

- **tag** (*str*) – the tag

- **filecontents** (*list[Tuple[str, str]]*) – a summary of the contents of the Zettels having this tag. The first component of the tuple is the path of the Zettel, the second may be its title.

to_xapian(*indexer: xapian.TermGenerator*) → xapian.Document
Export a Zettel as a new Xapian document

Parameters **indexer** (*xapian.TermGenerator*) – a TermGenerator

Returns A new Xapian document built from the data of the Zettel.

6.1.3 Zettelkasten

class xettel.base.Zettelkasten.**Zettelkasten**(*zettel_impl: Optional[Type[xettel.base.Zettelkasten.Z]] = None*)
Abstraction for a Zettelkasten folder

Keyword Arguments

- **zettel_impl** (*Optional[Type[ZettelType], None]*) – a class
- **file.** (*implementing a generating a Zettel from a*) –

folder
the path to the folder where Zettels are stored.

Type str

zettel_impl
cf. the argument above.

Type Optional[Type[ZettelType], None]

extension
the extension of the Zettels.

Type str

add_zettel_from_file(*filepath: str*) → None
Add Zettel within the Zettelkasten from an existing file, eventually replacing already existing Zettel with same UID

Parameters **filepath** (*str*) – the path to the Zettel

classmethod from_folder(*folder: str, zettel_impl: Type[xettel.base.Zettelkasten.Z]*) → Tuple[*xettel.base.Zettelkasten.Zettelkasten*[xettel.base.Zettelkasten.Z], list[typing.Tuple[int, int]]]
Class method for building a Zettelkasten for files in a folder

This method assumes all non-hidden files within a given folder are zettels.

It returns a tuple (ZK, List) consisting of the Zettelkasten and a list of unresolved links.

Parameters

- **folder** (*str*) – the path to the Zettelkasten
- **zettel_impl** – (Type[ZettelType]): a concrete subclass of ZettelFile.

Returns The Zettelkasten built from the folder together with a list of unresolved links.

Raises **TypeError** – if `zettel_impl` is not a concrete subclass of ZettelFile.

6.2 Implementations

6.2.1 Markdown (basic)

class xettel.impl.markdown.ZettelMD.**ZettelMD**(*parent: Zettelkasten*, *filename: str*)

Example class for markdown-formatted zettels.

Such a Zettel may have a Jekyll-style YAML frontmatter, as compatible with pandoc.

Links are assumed to be of the format '[#$UID]' where $UID is either a 12-char long numering string or a 8-char long alphanumeric string.

Files are expected to end in '.md'.

static export(*zettel:* xettel.base.ZAbstract.ZettelAbstract, *force: bool = False*) → None

Export the Zettel to HTML

get_indexing_function(*key: str*) → Callable[[xapian.Document, xapian.TermGenerator, Any], None]

Returns the function needed to index data from the attributes given a lowered and whitespace-stripped (as in str.strip()) key.

Functions returned by this method must have the following signature: `(xapian.Document, xapian.TermGenerator, value) -> None`.

Parameters **key** (*str*) – the key to the indexing function

Returns The indexing function corresponding to the key.

static issue_export_cmd(*source: str*, *dest: str*, *env: dict[str, str]*, *zinfo: dict[str, str]*) → None

Execute the command to export Zettels.

The abstract method defined here does nothing.

Parameters

- **source** (*str*) – path of the Zettel
- **dest** (*str*) – where to export the Zettel
- **env** (*dict[str, str]*) – dictionnary of environement variables to pass to the command.
- **zinfo** (*dict[str,str]*) – additional information to feed the command, probably directly among the arguments of the command.

parse_attributes() → None

Parses attributes from the file.

parse_links(*links=[]*) → None

Parses links in the file, according to the format specified above.

Keyword Arguments **links** (*Collection[int]*) – unused argument, here so that typechecking passes.

classmethod set_queryparser_prefixes(*queryparser: xapian.QueryParser*) → None

Sets the prefixes for searching the metadata defined by this class into the queryparser.

The method defined here does nothing.

Parameters **queryparser** (*xapian.QueryParser*) – the queryparser to feed information to.

classmethod tag_export(*tag: str*, *filecontents: list[typing.Tuple[str, str]]*) → None

Exports a page of a list of Zettels having this tag.

The abstract method defined here does nothing.

Parameters

- **tag** (*str*) – the tag
- **filecontents** (*list[Tuple[str, str]]*) – a summary of the contents of the Zettels having this tag. The first component of the tuple is the path of the Zettel, the second may be its title.

6.2.2 Xapian

ZettelX

class xettel.impl.xapian.ZettelX.**ZettelX**(*parent: ZettelkastenX*, *doc: xapian.Document*, *id: int*)

Parameters

- **parent** (ZettelkastenX) – the parent Zettelkasten to this Zettel
- **doc** (*xapian.Document*) – the Xapian document from which Xettel will reconstruct the Zettel
- **id** (*int*) – Xapian id of the document.

__xdata
data retrieved from Xapian

Type dict[str, Any]

__xapian_id
the id of the Xapian document.

Type int

parse_attributes() → None
Along with setting attributes, this methods also set backlinks

parse_links(*links: Collection[int] = []*) → None
Parse links with the given data

The abstract method defined here just sets the private attribute `__links` to the ones given by the argument `links`.

Parameters **links** (*Collection[int]*) – a collection of ints.

ZettelkastenX

class xettel.impl.xapian.ZettelkastenX.**ZettelkastenX**(*folder: str*)
Rebuilds a Zettelkasten from a xapian database.

Parameters **folder** (*str*) – path to the folder where Zettel files are stored.

classmethod **from_db**(*db: xapian.Database*, *folder: str*) → *xettel.impl.xapian.ZettelkastenX.ZettelkastenX*
Constructs a Zettelkasten from the whole database.

Parameters

- **db** (*xapian.Database*) – the Xapian database from which xettel will retreive the Zettels.
- **folder** (*str*) – path to the folder where Zettel files are stored.

Returns A completely reconstructed Zettelkasten

classmethod from_mset(*mset: xapian.MSet*, *folder: str*) → *xettel.impl.xapian.ZettelkastenX.ZettelkastenX*
Constructs a partial Zettelkasten with MSet matches.

Parameters

- **mset** (*xapian.MSet*) – a MSET from which xettel will reconstruct a Zettelkasten.
- **folder** (*str*) – path to the folder where Zettel files are stored.

Returns A partially reconstructed Zettelkasten

initialise_contents(*setbacklinks: bool = False*) → None
Same as parent's method, but does not initialise backlinks globalwise as it is normally already stored in the xapian database.

Keyword Arguments **setbacklinks** (*bool*) – whether to set backlinks or not

6.3 Searching

class xettel.search.**Search**(*reader: ZXReader*, *query: str*, *sort: bool = False*, *n: Optional[int] = None*)
The Search class provides an interface for common search operations.

Parameters

- **reader** (ZXReader) – a reader for the Xapian database.
- **query** (*str*) – a querystring for Xapian to parse. If the provided query is `'*'`, then rather than searching using Xapian, the whole Zettelkasten is retrieved.

reader
the reader.

Type *ZXReader*

query
the query.

Type str

sort
whether to sort by most recent

Type bool

n
the number of search results to ouput, `None' for all the results

Type Optional[int]

results
a SearchList object corresponding to the query.

Type *SearchList*

class xettel.search.**SearchList**(*zk: ZettelkastenAbstract*, *reader: ZXReader*)
The SearchList class provides a way to retreive information from a Xapian search.

Parameters

- **zk** (ZettelkastenAbstract) – a Zettelkasten
- **reader** (ZXReader) – a reader into the Xapian database

zettelkastenk
a Zettelkasten

Type *ZettelkastenAbstract*

reader
a reader into the Xapian database

Type *ZXReader*

get_results() → Iterator[*xettel.search.SearchResult*]
Iterator over the results of the query.

Yields Results from the query.

map() → dict[str, str]
Gives a mapping uid -> filename.

Returns A dictionnary mapping uids (as string, both in base 10 or in base36) to the corresponding filename.

output(*mode: str*) → str
Provides a string representing the results according to a chosen representation.

Parameters **mode** (*str*) – A string representing a mode. The modes currently implemented are:

- normal: gives a summary of each zettel containing its uid, its tags, its title and its abstract.
- uid: gives the uids in base 36
- tags: gives the tags appearing in the Zettels
- filename: gives the filename of each Zettel
- backlinks: gives the backlinks for each zettel
- map: gives a json-formatted map as described in SearchList.map
- json: gives a full json with all info and unexpanded uids

Raises **NotImplementedError** – if the mode is one of the above.

Returns A string whose lines are as described in the specification for the `mode` argument.

tags() → list[str]
Gives the tags that appear in the Zettels matching the query.

Returns A lexically ordered list with unique elements giving the tags of the zettels.

to_dict() → dict[str, dict[str, typing.Any]]
Gives a plain list of the results

Returns A dict of all the search results, indexed by their base 36 uid, without their text.

class xettel.search.**SearchResult**(*zettel: ZA*)
The class SearchResult provides an interface (coded as a MutableMapping) to get results from a Zettel.

The keys currently implemented with a special behaviour are:

- uid
- uid36
- filename
- backlinks

If they doesn't belong to the list above, then the value is taken from the Zettel's attributes dictionnary.

Specification for the special behaviour are explained in the method used to give them.

Parameters **zettel** (*ZA*) – a Zettel

item
a Zettel

Type ZA

backlinks(*zk: ZettelkastenAbstract*) → str
Gives a comma-separated list of the basenames of the Zettel's backlinks.

Returns a comma-separated list of the basenames of the Zettel's backlinks.

Return type str

filename() → str
Gives the filename of the Zettel

Returns A string with the filename.

format(*string: str*) → str
Takes advantages of the fact that this class is a mutable mapping to format a string with the attributes of the Zettel.

Returns A formatted string.

Return type str

normal() → str
Gives the default representation of a Zettel.

Returns abstract"

Return type A string formatted as "uid36 (#tags) – title

uid36() → str
Gives the uid in base 36.

Returns A string with the uid in base 36.

class xettel.search.**SetEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

default(*obj*)
Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

6.4 Graph-plotting module

class xettel.graph.**ZKGraph**

Class for handling graph-related operation on a Zettelkasten

The __init__ method only initialises self.nxgraph.

nxgraph
networkx graph built from a Zettelkasten

pgvgraph
pygraphviz graph built from self.nxgraph

zk
Zettelkasten from which the nxgraph is built.

draw_graph(*path: str*) → None

Draws the graph using graphviz to the specified path with desired extension

Parameters **path** (*str*) – the path to write to.

export_to_cytoscape() → str
Exports a JSON dictionnary for use with Cytoscape.js

classmethod from_zettelkasten(*zk: ZettelkastenAbstract*) → *ZKGraph*
Builds a networkx graph from an initialised zettelkasten

Parameters **zk** (*ZettelkastenAbstract*) – a Zettelkasten

Returns An instance of this class with self.nxgraph properly initialised.

initialise_graphviz() → None
Converts networkx graph to a pygraphviz graph

initialise_nx_graph_from_zk() → None
Initialises a networkx graph from a Zettelkasten. Assumes that the Zettelkasten has been properly initialised.

is_connected() → bool
Tests is graph fron Zettelkasten is weakly connected. Assumes that the nx graph has already been initialised.

Returns A boolean, True if the nxgraph is weakly connected.

set_graphviz_labels() → None
Sets labels to graphviz graph for export

set_graphviz_layout(*program: str = 'neato'*, *model: str = 'circuit'*) → None
Runs the graphviz's program to lay out the nodes accordingly

Keyword Arguments

- **program** (*str*) – the program to use
- **model** (*str*) – the model to use

set_graphviz_layout_attributes() → None
Sets attributes to individual edges and nodes prior to running the layout

unconnected_zettels_uids() → set[int]
Returns a set of zettels uid which are not in the same weakly connected component as '0'.

Returns A set of uids of Zettels not connected to the root.

xettel.graph.**name**(*uid: int*, *zk: ZettelkastenAbstract*) → str
Extract raw name of the zettel

Parameters

- **uid** (*int*) – uid of the Zettel.
- **zk** (ZettelkastenAbstract) – Zettelkasten

Returns Basename of the Zettel

xettel.graph.**url**(*uid: int*, *zk: ZettelkastenAbstract*) → str
Gives the url the graph's nodes should point to

Parameters

- **uid** (*int*) – uid of the Zettel
- **zk** (ZettelkastenAbstract) – Zettelkasten

Returns Path to the export url of the Zettel (only html implemented so far).

6.5 Interfacee to *xapian*

6.5.1 Common interface for IO

class xettel.zxapian.basic.**ZX**(*folder: str*, *db: Optional[xapian.Database] = None*)
Common interface for reading and writing the Xapian database

6.5.2 Reader Class

class xettel.zxapian.reader.**ZXReader**(*folder: str*)
Class for reading the Xapian database

add_prefix(*db_prefix: str*, *user_prefix: str*) → None
Add a synonym for a prefix

db_to_zk() → *xettel.impl.xapian.ZettelkastenX.ZettelkastenX*
Export the database to a Zettelkasten

search(*querystring: str*, *sort: bool = False*, *n: Optional[int] = None*) → xapian.MSet
Search the database (with limitations) Basically no wrapper around Xapian

The method also implements a different way of querying the full database when `querystring` is `'*'` than `ZXReader.db_to_zk`. Instead of rebuilding the Zettelkasten, this returns a `xapian.MSet`. This allows for an efficient sorting which is done by `xapian`, as well as a limited subset of the results.

search_zk(*querystring: str*, *sort: bool = False*, *n: Optional[int] = None*) →
xettel.impl.xapian.ZettelkastenX.ZettelkastenX
Search the database and returns a Zettelkasten with only the matching zettels.

tag_map() → Iterator[Tuple[str, list[typing.Tuple[str, str]]]]
Returns a list [(tagname, [filenames_with_that_tag])]. Uses the resources module to determine the tag prefix.

6.5.3 Writer Class

class xettel.zxapian.writer.**ZXWriter**(*folder: str*)

delete_in_database(*zettelkasten: Zettelkasten[ZFT]*, *stored_zk: Optional[ZettelkastenX[ZFT]] = None*) → int

Delete Zettels not present in provided Zettelkasten from the db

Returns the number of deleted zettels

index_zettel(*zettel: xettel.zxapian.writer.ZFT*) → None

Indexes a single zettel into the xapian database.

Given that the indexing strategy is to map ZK UIDs, this eventually overwrites a previous zettel indexed with the same UID.

update_database(*zettelkasten: Zettelkasten[ZFT]*, *delete: bool = True*) → Tuple[int, int, int]

Given a Zettelkasten, replace index zettels whose timestamp is older. Optionally deletes zettels not present in provided Zettelkasten (defaults to True).

Returns a 3-uple of ints (updated, added, deleted)

zk_to_db(*zettelkasten: Zettelkasten*, *delete: bool = True*) → int

Indexes a whole Zettelkasten into the xapian database, overwriting everything.

Optionally deletes zettels that don't exist anymore (defaults to True)

Returns the number of deleted documents.

6.5.4 Misc.

6.6 Implementations Register

xettel.implRegister.**register_Z**(*name: str*) → Callable[[Type[ZettelFile]], Type[ZettelFile]]

Decorator for classes that registers Zettel implementations.

This decorator only adds the decorated class into the dictionary `Z_REGISTRY` at the name provided and does nothing else.

Classes decorated with this decorator are expected to be subclasses of `xettel.base.ZettelFile.ZettelFile` but no runtime typechecking is done.

See also:

One instance of use of this decorator is the class `xettel.impl.markdown.ZettelMD.ZettelMD` which may serve as an example for building your own implementation.

Example

Another example is the class defined in `doc/config/impl/myMD.py`. It is the class I personally use and provides additional preprocessing to that of `pandoc` which allows for the use of `tikz-cd` for instance.

Parameters **name** (*str*) – the name under which register the class.

Returns A function registering the class and returning the said class.

`xettel.implRegister.`**register_ZK**(*name: str*) → Callable[[Type[Zettelkasten]], Type[Zettelkasten]]
Decorator for classes that registers Zettelkasten implementations.

This decorator only adds the decorated class into the dictionary `ZK_REGISTRY` at the name provided and does nothing else.

Classes decorated with this decorator are expected to be subclasses of `xettel.base.Zettelkasten.Zettelkasten` but no runtime typechecking is done.

Parameters **name** (*str*) – the name under which register the class.

Returns A function registering the class and returning the said class.

6.7 CLI Interface

6.7.1 Configuration handling

This module provides functions for retreiving the configuration from the config file and loading the end-user's implementations of a Zettelkasten and a Zettel.

`xettel.config.`**config**
a dictionnary corresponding to the user's configuration.

Type MutableMapping[str, Any]

`xettel.config.`**config_path**
path to the config file

Type str

`xettel.config.`**config_dir**
path to the config directory

Type str

`xettel.config.`**ZK_PATH**
path to the Zettelkasten directory

Type str

`xettel.config.`**ZK_CLASS**
a subclass of `xettel.base.Zettelkasten.Zettelkasten` chosen by the user as the class to use for managing the Zettelkasten.

Type Type[*Zettelkasten*]

`xettel.config.`**Z_CLASS**
a subclass of `xettel.base.ZettelFile.ZettelFile` chosen by the user to use for managing individual Zettels.

Type Type[Zettel]

xettel.config.**load_implementations**() → None
Loads user defined classes. Must be put in the directory "impl" of the config directory.

xettel.config.**pursue_read_command_check**() → None
Errors if the database cannot be queried and prints a nice message, nop otherwise.

xettel.config.**set_config**(*configpath: str*, *zk_dir: Optional[str] = None*, *xettel_dir: Optional[str] = None*) → None
Sets the config and the config_path variable of this module

Parameters **configpath** (*str*) – the path to the config file.

Keyword Arguments **zk_dir** (*Optional[str]*) – the Zettelkasten directory

xettel.config.**set_implementation**() → None
Sets the Z_CLASS and ZK_CLASS variables of this module

xettel.config.**zk_dir_to_zk_data_dir**(*zk_dir: str*) → str
Translates a path to the ZK files to the path where the xapian db and all other files are stored.

6.7.2 Commands

6.8 Shared functions

This modules provides diverse functions used throughout the project.

xettel.shared.**int36**(*num: int*) → str
Converts an int to a base36 uppercase representation.

Parameters **num** (*int*) – a number

Returns An uppercase base 36 representation.

Return type str

CHAPTER

SEVEN

NEOVIM PLUGIN

For convenience, alongside the cli program, I've provided a tiny Neovim plugin that works with subclasses of *ZettelMD*.

To install it, copy the files in the folder `extra/vim` to your Neovim config folder, execute `:UpdateRemotePlugins` in Neovim, and replace the path in `ftdetect/zk.vim` and in `xettel.vim` by the path to your Zettelkasten folder.

In `xettel.vim` are defined some keybindings. In particular, entering `<CR>` on top of a link opens it, `<BS>` opens a backlink, `<Leader><CR>` creates a new Zettel and inserts the link in the current file and `<Leader>#` insert the link of a Zettel matching a query.

7.1 Commands

The plugin exports the following commands:

`:XettelOpenLink` Open the link under the cursor

`:XettelOpenBacklinks` Chose a backlink and open it.

`:XettelNewZettel` Interactively create a new Zettel.

CHAPTER

EIGHT

CONTRIBUTING

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

INDEX

Symbols

A

N

O

P

Q

R

S

T

U

X

Z