Bibolamazi Documentation

Release 3.0

Philippe Faist

May 31, 2015

Contents

Table of Contents:

CHAPTER 1

Introduction to Bibolamazi

Bibolamazi lets you prepare consistent and uniform BibTeX files for your LaTeX documents. It lets you prepare your BibTeX entries as you would like them to be—adding missing or dropping irrelevant information, capitalizing names or turning them into initials, converting unicode characters to latex escapes, etc.

1.1 Example Usage Scenario

A typical scenario of Bibolamazi usage might be:

- You use a bibliography manager, such as Mendeley, to store all your references. You have maybe configured e.g. Mendeley to keep a BibTeX file `Documents/bib/MyLibrary.bib` in sync with your library;
- You're working, say on a document `mydoc.tex`, which cites entries from `MyLibrary.bib`;
- You like to keep URLs in your entries in your Mendeley library, because it lets you open the journal page easily, but you don't want the URLs to be displayed in the bibliography of your document `mydoc.tex`. But you've gone through all the bibliography styles, and really, the one you prefer unfortunatly does display those URLs.
- You don't want to edit the file `MyLibrary.bib`, because it would just be overwritten again the next time you open Mendeley. The low-tech solution (what people generally do!) would then be to export the required citations from Mendeley to a new bibtex file, or copy `MyLibrary.bib` to a new file, and edit that file manually.
- To avoid having to perform this tedious task manually, you can use Bibolamazi to prepare the BibTeX file as you would like it to be. For this specific task, for example, you would perform the following steps:
  - Create a bibolamazi file, say, `mydoc.bibolamazi.bib`;
  - Specify as a source your original `MyLibrary.bib`:

```
src: ~/Documents/bib/MyLibrary.bib
```

  - Give the following filter command:

```
filter: url -dStrip
```

    which instructs to strip all urls (check out the documentation of the *url* filter in the *Help & Reference Browser*)

  - Run bibolamazi.
  - Use this file as your bibtex bibliography, i.e. in your LaTeX document, use:

```
\bibliography{mydoc.bibolamazi}
```

Note that you can then run Bibolamazi as many times as you like, to update your file, should there have been changes to your original `MyLibrary.bib`, for example.

1.2 Teaser: Features

The most prominent features of Bibolamazi include:

- A *duplicates* filter allows you to efficiently collaborate on LaTeX documents: in your shared LaTeX document, each collaborator may cite entries in his own bibliography database (each a source in the bibolamazi file). Then, if instructed to do so, bibolamazi will detect when two entries are duplicates of each other, merge their information, and produce LaTeX definitions such that the entries become aliases of one another. Then both entry keys will refer to the same entry in the bibliography.

  Catch: there is one catch to this, though, which we can do nothing about: if two entries in two different database share the same key, but refer to different entries. This may happen, for example, if you have automatic citation keys of the form `AuthorYYYY`, and if the author published several papers that same year.

- A powerful *arxiv* filter, which can normalize the way entries refer to the arXiv.org online preprint repository. It can distinguish between published and unpublished entries, and its output is highly customizable.

- A general-purpose *fixes* filter provides general fixes that are usually welcome in a BibTeX files. For example, revtex doesn't like Mendeley's way of exporting swedish 'Å', for example in `Åberg`, as `\AA berg`, and introduces a space between the 'Å' and the 'berg'. This filter allows you to fix this.

- Many more! Check out the filter list in the *Help & Reference Browser* window of Bibolamazi!

CHAPTER 2

Downloading and Installing Bibolamazi

Bibolamazi comes in two flavors:

- an Application that runs on Mac OS X, Linux and Windows (this is what most users probably want)
- a command-line tool (for more advanced and automated usage)

There are precompiled ready-for-use binaries for the Application (see below, *The Bibolamazi Application*). Alternatively, both flavors may be installed using `pip`/`setuptools` or from source (see *Installing the Command-Line Interface*).

2.1 The Bibolamazi Application

If you're unsure which flavor to get, this is the one you're looking for. It's straightfoward to download, there is no installation required, and the application is easy to use.

Download the latest release from our releases page:

Download Release: https://github.com/phfaist/bibolamazi/releases

These binaries don't need any installation, you can just download them, place them wherever you want, and run them.

You may now start using Bibolamazi normally. To read more on bibolamazi, skip to *Using the Bibolamazi Application*.

2.2 Installing the Command-Line Interface

Bibolamazi runs with Python 2.7 (this is there by default on most linux and Mac systems).

Additionally, the graphical user interface requires PyQt4. If you're on a linux distribution, it's most probably in your distribution packages. Note you only need PyQt4 to run the graphical user interface: the command-line version will happily run without.

The easy way: via PIP

The recommended way to install Bibolamazi command line and gui interfaces is via `pip`:

```
pip install bibolamazi        # for the command-line interface
pip install bibolamazigui     # if you want the GUI interface
```

After that, you'll find the `bibolamazi` (respectively `bibolamazi_gui`) executables in your PATH:

```
> bibolamazi --help           # command-line interface
(...)
> bibolamazi_gui              # to launch the GUI
(...)
```

The less easy way: From Source

You may, alternatively, download and compile the packages from source.

- First, clone this repository on your computer (don't download the prepackaged ZIP/Tarball proposed by github, because there will be missing submodules):

```
> cd somewhere/where/Ill-keep-bibolamazi/
...> git clone --recursive https://github.com/phfaist/bibolamazi
```

  Note the `--recursive` switch which will also retrieve all required submodules.

- Then, run the setup script to install the package and script (see Installing Python Modules):

```
> python setup.py install
```

  After that, you should find the `bibolamazi` executable in your PATH automatically:

```
> bibolamazi --help
```

- If you want to install the GUI Application, you need to do that seperately. Go into the `gui/` directory of the source code, and run the python setup script there:

```
> cd gui/
gui> python setup.py install
```

  After that, you should find the `bibolamazi_gui` executable in your PATH automatically:

```
> bibolamazi_gui
```

CHAPTER 3

Using the Bibolamazi Application

3.1 Bibolamazi Operating Mode

Bibolamazi works by reading your reference bibtex files—the 'sources', which might for example have been generated by your favorite bibliography manager or provided by your collaborators—and merging them all into a new file, applying specific rules, or 'filters', such as turning all the first names into initials or normalizing the way arxiv IDs are presented.

The Bibolamazi file is this new file, in which all the required bibtex entries will be merged. When you prepare you LaTeX document, you should create a new bibolamazi file, and provide that bibolamazi file as the bibtex file for the bibliography.

When you open a bibolamazi file, you will be prompted to edit its configuration. This is the set of rules which will tell bibolamazi where to look for your bibtex entries and how to handle them. You first need to specify all your sources, and then all the filters.

The bibolamazi file is then a valid BibTeX file to include into your LaTeX document, so if your bibolamazi file is named `main.bibolamazi.bib`, you would include the bibliography in your document with a LaTeX command similar to:

```
\bibliography{main.bibolamazi}
```

3.2 The Bibolamazi Configuration Section

If you open the Bibolamazi application and open your bibolamazi file (or create a new one), you'll immediately be prompted to edit its configuration section.

3.2.1 Specifying sources

Sources are where your 'original' bibtex entries are stored, the ones you would like to process. This is typically a bibtex file which a reference manager such as Mendeley keeps in sync.

Sources are specified with the `src:` keyword. As an example:

```
% src: mysource.bib
```

You should specify one or more files from which entries should be read. If more than one file is given, only the FIRST file that exists is read. This is useful for example, if on different computers your bibtex is elsewhere:

```
% src: /home/philippe/bibtexfiles/mylibrary.bib /Users/philippe/bibtexfiles/mylibrary.bib
```

You may also specify HTTP or FTP URLs. If your filename or URL contains spaces, enclose the name in double quotes: `"My Bibtex Library.bib"`.

To specify several sources that should be read independently, simply use multiple `src:` commands:

```
% src: file1.bib [alternativefile1.bib ...]
% src: file2.bib [alternativefile2.bib ...]
% [...]
```

This would collect all the entries from the first existing file of each `src:` command.

3.2.2 Specifying filters

Once all the entries are collected from the various sources, you may now apply filters to them.

A filter is applied using the `filter:` command:

```
% filter: filtername [options and arguments]
```

Filters usually accept options and arguments in a shell-like fashion, but this may vary in principle from filter to filter. For example, one may use the *arxiv* filter to strip away all arXiv preprint information from all published entries, and normalize unpublished entries to refer to the arxiv in a uniform fashion:

```
% filter: arxiv --mode=strip --unpublished-mode=eprint
```

A full list of options can be obtained with:

```
> bibolamazi --help arxiv
```

and more generally, for any filter:

```
> bibolamazi --help <filtername>
```

A list of available filters can be obtained by running:

```
> bibolamazi --list-filters
```

Note: Filters are organized into *filter packages* (see below). A filter is searched in each filter package until a match is found. To force the lookup of a filter in a specific package, you may prefix the package name to the filter, e.g.:

```
% filter: myfilterpackage:myfiltername --option1=val1  ...
```

3.3 Example/Template Configuration Section

```
%% BIBOLAMAZI configuration section.
%% Additional two leading percent signs indicate comments in the configuration.

%% **** SOURCES ****

%% The _first_ accessible file in _each_ source list will be read and filtered.

src:   <source file 1> [ <alternate source file 1> ... ]
src:   <source file 2> [ ... ]

%% Add additional sources here. Alternative files are useful, e.g., if the same
```

```
%% file must be accessed with different paths on different machines.

%% **** FILTERS ****

%% Specify filters here. Specify as many filters as you want, each with a `filter:'
%% directive. See also `bibolamazi --list-filters' and `bibolamazi --help <filter>'.

filter: filter_name  <filter options>

%% Example:
filter: arxiv -sMode=strip -sUnpublishedMode=eprint

%% Finally, if your file is in a VCS, sort all entries by citation key so that you don't
%% get huge file differences for each commit each time bibolamazi is run:
filter: orderentries
```

3.4 Available Filters

You can get a full list of available filters if you open the bibolamazi help & reference browser window (from the main application startup window). You can click on the various filters displayed to view their documentation on how to use them.

3.5 Filter Packages

Filters are organized into *filter packages*. All built-in filters are in the package named *filters*. If you want to write your own filters, or use someone else's own filters, then you can install further filter packages.

A *filter package* is a Python package, i.e. a directory containing a `__init__.py` file, which contains python modules that implement the bibolamazi filter API.

If you develop your own filters, it is recommended to group them in a filter package, and not for example fiddle with the built-in filter package. Put your filters in a directory called, say, *myfilters*, and place an additional empty file in it called *__init__.py*. This will create a python package named *myfilters* with your filters as submodules.

To register the filter packages so that bibolamazi knows where to look for your filters, open the settings dialog, and click "Add filter package ..."; choose the directory corresponding to your filter package (e.g. *myfilters*). Now you can refer in your bibolamazi file to the filters within your filter package with the syntax `myfilters:filtername` or simply `filtername` (as long as the filter name does not clash with another filter of the same name in a different filter package).

CHAPTER 4

Using Bibolamazi in Command-Line

4.1 First Steps With Bibolamazi Command-Line

Once you've installed bibolamazi as described in *Installing the Command-Line Interface*, you may start using it! Here are a couple of commands to get you started playing around. But it's important to understand how Bibolamazi works: for that, read the following sections of this manual carefully.

- To compile a bibolamazi bibtex file, you should run `bibolamazi` in general as:

```
> bibolamazi bibolamazibibtexfile.bibolamazi.bib
```

- To quickly get started with a new bibolamazi file, the following command will create the given file and produce a usable template which you can edit:

```
> bibolamazi --new newfile.bibolamazi.bib
```

- For an example to study, look at the test files `test/testX.bibolamazi.bib` in the source code. To compile them, run:

```
> bibolamazi test/test0.bibolamazi.bib
```

- For a help message with a list of possible options, run:

```
> bibolamazi --help
```

  To get a list of all available filters along with their description, run:

```
> bibolamazi --list-filters
```

  To get information about a specific filter, simply use the command:

```
> bibolamazi --help <filter>
```

4.2 Bibolamazi Operating Mode

Bibolamazi works by reading a bibtex file (say `main.bibolamazi.bib`) with a special bibolamazi configuration section at the top. These describe on one hand *sources*, and on the other hand *filters*. Bibolamazi first reads all the entries in the given sources (say `source1.bib` and `source2.bib`), and then applies the given filters to them. Then, the main bibtex file (in our example `main.bibolamazi.bib`) is updated, such that:

- Any content that was already present in the main bibtex file *before* the configuration section is restored unchanged;

- The configuration section is restored as it was;
- All the filtered entries (obtained from, e.g., `source1.bib` and `source2.bib`) are then dumped in the rest of the file, overwriting the rest of `main.bibolamazi.bib` (which logically contained output of a previous run of bibolamazi).

The bibolamazi file `main.bibolamazi.bib` is then a valid BibTeX file to include into your LaTeX document, so you would include the bibliography in your document with a LaTeX command similar to:

```
\bibliography{main.bibolamazi}
```

4.3 The Bibolamazi Configuration Section

The main bibtex file should contain a block of the following form:

```
%%%-BIB-OLA-MAZI-BEGIN-%%%
%
%   ... bibolamazi configuration section ...
%
%%%-BIB-OLA-MAZI-END-%%%
```

The configuration section is started by the string *%%%-BIB-OLA-MAZI-BEGIN-%%%* on its own line, and is terminated by the string *%%%-BIB-OLA-MAZI-END-%%%*, also on its own line. The lines between these two markers are the body of the configuration section, and are where you should specify sources and filters. Leading percent signs on these inner lines are ignored. Comments can be specified in the configuration body with two additional percent signs, e.g.:

```
% %% This is a comment
```

4.4 Content of the Configuration Section

The content of the configuration section is the same as described in *The Bibolamazi Configuration Section*. Of course, you'll probably want to prefix all lines by an additional '%' to make sure it gets interpreted as a bibtex comment (see example below).

4.5 Example Full Bibolamazi File

Here is a minimal example of a bibolamazi bibtex file:

```
.. Additionnal stuff here will not be managed by bibolamazi. It will also not be
.. overwritten. You can e.g. temporarily add additional references here if you
.. don't have bibolamazi installed.


%%%-BIB-OLA-MAZI-BEGIN-%%%
%
% %% BIBOLAMAZI configuration section.
% %% Additional two leading percent signs indicate comments in the configuration.
%
% %% **** SOURCES ****
%
% %% The _first_ accessible file in _each_ source list will be read and filtered.
%
```

```
% src:   <source file 1> [ <alternate source file 1> ... ]
% src:   <source file 2> [ ... ]
%
% %% Add additional sources here. Alternative files are useful, e.g., if the same
% %% file must be accessed with different paths on different machines.
%
% %% **** FILTERS ****
%
% %% Specify filters here. Specify as many filters as you want, each with a `filter:'
% %% directive. See also `bibolamazi --list-filters' and `bibolamazi --help <filter>'.
%
% filter: filter_name  <filter options>
%
% %% Example:
% filter: arxiv -sMode=strip -sUnpublishedMode=eprint
%
% %% Finally, if your file is in a VCS, sort all entries by citation key so that you don't
% %% get huge file differences for each commit each time bibolamazi is run:
% filter: orderentries
%
%%%-BIB-OLA-MAZI-END-%%%
%
%
% ALL CHANGES BEYOND THIS POINT WILL BE LOST NEXT TIME BIBOLAMAZI IS RUN.
%

... bibolamazi filtered entries ...
```

4.6 Querying Available Filters and Filter Documentation

A complete list of available filters, along with a short description, is obtained by:

```
> bibolamazi --list-filters
```

Run that command to get an up-to-date list. At the time of writing, the list of filters is:

```
> bibolamazi --list-filters

List of available filters:
--------------------------

Package `filters':

  arxiv         ArXiv clean-up filter: normalizes the way each biblographic
                entry refers to arXiv IDs.
  citearxiv     Filter that fills BibTeX files with relevant entries to cite
                with \cite{1211.1037}
  citekey       Set the citation key of entries in a standard format
  duplicates    Filter that detects duplicate entries and produces rules to make
                one entry an alias of the other.
  fixes         Fixes filter: perform some various known fixes for bibtex
                entries
  nameinitials  Name Initials filter: Turn full first names into only initials
                for all entries.
  only_used     Filter that keeps only BibTeX entries which are referenced in
                the LaTeX document
```

```
  orderentries  Order bibliographic entries in bibtex file
  url           Remove or add URLs from entries according to given rules, e.g.
                whether DOI or ArXiv ID are present

--------------------------

Filter packages are listed in the order they are searched.

Use  bibolamazi --help <filter>  for more information about a specific filter
and its options.
```

4.7 Specifying Filter Packages

The command-line bibolamazi by default only knows the built-in fitler package `filters`. You may however specify additional packages either by command-line options or with an environment variable.

You can specify additional filter packages with the command-line option `--filter-package`:

```
> bibolamazi myfile.bibolamazi.bib --filter-package 'package1=/path/to/filter/pack'
```

The argument to `--filter-package` is of the form 'packagename=/path/to/the/filter/package'. Note that the path is which path must be added to python's `sys.path` in order to import the `filterpackagename` package itself, i.e. the last item of the path must not be the package directory.

This option may be repeated several times to import different filter packages. The order is relevant; the packages specified last will be searched for first.

You may also set the environment variable `BIBOLAMAZI_FILTER_PATH`. The format is `filterpack1=/path/to/somewhere:filterpack2=/path/to/otherplace:...`, i.e. a list of filter package specifications separated by ':' (Linux/Mac) or ';' (Windows). Each filter package specification has the same format as the command-line option argument. In the environment variable, the first given filter packages are searched first.

CHAPTER 5

Writing a New Filter

5.1 Example of a custom filter

```
import random # for example purposes

# use this for logging output
import logging
logger = logging.getLogger(__name__)

# core filter classes
from bibolamazi.core.bibfilter import BibFilter, BibFilterError
# types for passing arguments to the filter
from bibolamazi.core.bibfilter.argtypes import CommaStrList, enum_class
# utility to parse boolean values
from bibolamazi.core.butils import getbool


# --- help texts ---

HELP_AUTHOR = u"""\
Test Filter by Philippe Faist, (C) 2014, GPL 3+
"""

HELP_DESC = u"""\
Test Filter: adds a 'testFilter' field to all entries, with various values.
"""

HELP_TEXT = u"""
There are three possible operating modes:

    "empty"  -- add an empty field 'testField' to all entries.
    "random" -- the content of the 'testField' field which we add to all entries
                is completely random.
    "fixed"  -- the content of the 'testField' field which we add to all entries
                is a hard-coded, fixed string. Surprise!

Specify which operating mode you prefer with the option '-sMode=...'. By
default, "random" mode is assumed.
"""

# --- operating modes ---
```

```
# Here we define a custom enumeration type for passing as argument to our
# constructor. By doing it this way, instead of simply accepting a string,
# allows the filter factory mechanism to help us report errors and provide more
# helpful help messages. Also, in the graphical interface the relevant option is
# presented as a drop-down list instead of a text field.

# numerical values -- numerical values just have to be different
MODE_EMPTY = 0
MODE_RANDOM = 1
MODE_FIXED = 2

# symbolic names and to which values they correspond
_modes = [
    ('empty', MODE_EMPTY),
    ('random', MODE_RANDOM),
    ('fixed', MODE_FIXED),
    ]

# our Mode type. See `bibolamazi.core.bibfilter.argtypes`
Mode = enum_class('Mode', _modes, default_value=MODE_NONE,
                  value_attr_name='mode')


# --- the filter object itself ---

class MyTestFilter(BibFilter):

    # import help texts above here
    helpauthor = HELP_AUTHOR
    helpdescription = HELP_DESC
    helptext = HELP_TEXT

    def __init__(self, mode="random", use_uppercase_text=False):
        """
        Constructor method for TestFilter.

        Note that this part of the constructor docstring itself isn't that
        useful, but the argument list below is parsed and used by the default
        automatic option parser for filter arguments. So document your
        arguments! If your filter accepts `**kwargs`, you may add more arguments
        below than you explicitly declare in your constructor prototype.

        If this function accepts `*args`, then additional positional arguments
        on the filter line will be passed to those args. (And not to the
        declared arguments.)

        Arguments:
          - mode(Mode): the operating mode to adopt
          - use_uppercase_text(bool): if set to True, then transform our added
            text to uppercase characters.
        """

        BibFilter.__init__(self)

        self.mode = Mode(mode)
        self.use_uppercase_text = getbool(use_uppercase_text)

        logger.debug('test filter constructor: mode=%s, use_uppercase_text=%s',
```

```
                    self.mode, self.use_uppercase_text)

    def action(self):
        # Here, we want the filter to operate entry-by-entry (so the function
        # `self.filter_bibentry()` will be called). If we had preferred to
        # operate on the whole bibliography database in one go (as, e.g., for
        # the `duplicates` filter), then we would have to return
        # `BibFilter.BIB_FILTER_BIBOLAMAZIFILE` here, and provide a
        # `filter_bibolamazifile()` method.
        #
        return BibFilter.BIB_FILTER_SINGLE_ENTRY

    def requested_cache_accessors(self):
        # return the requested cache accessors here if you are using the cache
        # mechanism.  This also applies if you are using the `arxivutil`
        # utilities.
        return [ ]

    def filter_bibentry(self, entry):
        #
        # entry is a pybtex.database.Entry object
        #

        if self.mode == MODE_EMPTY:
            entry.fields['testField'] = ''

        elif self.mode == MODE_RANDOM:
            entry.fields['testField'] = random.randint(0, 999999)

        elif self.mode == MODE_FIXED:
            entry.fields['testField'] = (
                u"On d\u00E9daigne volontiers un but qu'on n'a pas "
                u"r\u00E9ussi \u00E0 atteindre, ou qu'on a atteint "
                u"d\u00E9finitivement. (Proust)"
                )
        else:
            raise BibFilterError('testfilter', "Unknown operating mode: %s"
                                 % mode )

        if self.use_uppercase_text:
            entry.fields['testField'] = entry.fields['testField'].toupper()

        return

#
# Every python module which defines a filter should have the following method,
# which returns the filter class type (which is expected to be a `BibFilter`
# subclass).
#
def bibolamazi_filter_class():
    return MyTestFilter
```

5.2 Developing Custom filters

Writing filters is straightforward. An example is provided here: *Example of a custom filter*. Look inside the `bibolamazi/filters/` directory at the existing filters for further examples, e.g. `arxiv.py`, `duplicates.py`

or `url.py`. They should be rather simple to understand.

A filter can either act on individual entries (e.g. the `arxiv.py` filter), or on the whole database (e.g. *duplicates.py*).

For your organization, it is recommended to develop your filter(s) in a custom filter package which you keep a repository e.g. on github.com, so that the filter package can be easily installed on the different locations you would like to run bibolamazi from.

Don't forget to make use of the *bibolamazi cache*, in case you fetch or compute values which you could cache for further reuse. You should access caches through the *BibUserCacheAccessor* class. Look at for the documentation for the *bibusercache* module. Look at examples most of all!! *(TODO: add documentation about caches)*

There are a couple utilities provided for the filters, check the `bibolamazi.filters.util` module. In particular check out the *arxivutil* and *auxfile* modules.

Feel free to contribute filters, it will only make bibolamazi more useful!

5.3 The Filter Module

There are two main objects your module should define at the very least:

- a filter class, subclass of *BibFilter*.
- a method called `bibolamazi_filter_class()`, which should return the filter class object. For example:

```
def bibolamazi_filter_class():
    return ArxivNormalizeFilter;
```

You may want to have a look at *Example of a custom filter* for an example of a custom filter.

Your filter should log error, warning, information and debug messages to a logger obtained via Python's logging mechanism, as demonstrated in the example.

5.4 Passing Arguments to the Filter

Command line arguments passed to the filter in the user's bibolamazi config section are parsed into Python arguments to the filter class' constructor. The translation is rather intuitive: each argument to the filter may be specified as an option, either using the syntax `--use-uppercase=value` or `--use-uppercase value`, where underscores are replaced by dashes, or using the Ghostscript-like syntax `-dUseUppercase` or `-dUseUppercase=false`, or for other types `-sMode=fixed`.

Some remarks:

- to each filter argument corresponds a command-line option starting with `--`, where underscores are replaced by dashes. The command-line takes a single mandatory argument (except for arguments declared as booleans in their arg-docs, see *Argdocs: Filter Argument Documentation* below).
- to each filter argument, corresponds a command-line option starting with `-d` or `-s`, using the syntax `-dFilterOptionName`, `-dFilterOptionName=Value` or `-sFilterOptionName=Value`. The `-d` variant is used to specify boolean option values, the `-s` variant any other type. The `FilterOptionName` is obtained by camel-casing the filter python argument: for example, if the filter constructor accepts an argument named `use_uppercase_chars`, then the corresponding camel-cased version will be `UseUppercaseChars`. (See note below on case sensitivity.)
- each filter argument may be documented using *Argdocs: Filter Argument Documentation*. This information will appear in the filter help text.

- if the filter constructor accepts a `**kwargs`, then any additional option-value pairs given as `-sKey=Value` or `-dKey` or `-dKey=Value` are passed on to the filter constructor's `kwargs`.
- if the filter constructor accepts a `*args`, then any additional positional arguments on the command line is passed to that `*args` parameter. The ordering of positional and optional arguments on the command-line make no difference. (Note that this also works this way if not all the previous declared arguments are specified. There's some python hacking in there ;))

Note: If even a single filter argument uses an uppercase letter, then the option parser will not convert any letter casing, and all option names will have the exact same letter casing as the filter arguments. Similarly, no camel-casing will occur with the `-s...` or `-d...` options.

5.5 Filter General Help Documentation

The filter class should declare the members `helpauthor`, `helpdescription` and `helptext` with meaningful help text:

- `helpauthor` should be a short one-line description of the filter and contributor with license. E.g.:

```
ArXiv clean-up filter by Philippe Faist, (C) 2013, GPL 3+
```

- `helpdescription` is a brief description of what the filter does. This is displayed right after the *Usage* section in the help text, and before the filter arguments description.
- `helptext` is a long description of what the filter exactly does, how to use it, the advantages, tricks, pitfalls, etc.

In the built-in filters, as well as the examples, the text is declared outside of the class (see `HELP_AUTHOR` etc.) so that we don't have to deal with the indentation (and in the class, we only have `helpauthor=HELP_AUTHOR` etc.). That's perfectly fair and completely optional.

5.6 Argdocs: Filter Argument Documentation

The docstring of the filter constructor is parsed in a special way. Documentation of the function arguments are specially parsed: they should have the form:

```
- argument_name(type): Description of the argument. The description may
  span over several lines.
- other_argument_name: Description of the other option. Notice that the
  type is optional and will default to a simple string.
```

This information will be displayed when running `bibolamazi --help filtername`.

If a *type* is specified, it should be a name of a python type, or a type which is available in the namespace of the filter module. The filter factory will attempt to convert the given string to the specified type when calling the filter constructor. If the given *type* is a custom type, and it has a docstring, then the docstring is included in the "Note on Filter Options Syntax" section of the help text.

There are some convenient predefined types for filter arguments, all defined in the module `bibolamazi.bibfilter.argtypes`:

- *`CommaStrList`*: a comma-separated list of strings. This type may directly be used as a list type.
- *`enum_class()`*: a function which returns a custom class which represents an enumeration value of several options.

Maybe look at the built-in filters and other examples to get an idea.

More doc should come here at some point in the future.....

5.7 Customizing Default Behavior

There are several other functions the module may define, although they are not mandatory.

- *parse_args()* should parse an argument string, and return a tuple `(args, kwargs)` of how the filter constructor should be called. If the module does not provide this function, a very powerful default automatic filter option processor (based on python's `argparse` module) is built using the filter argument names as options names.
- *format_help()* should return a string with full detailed information about how to use the filter, and which options are accepted. If the module does not provide this function, the default automatic filter option processor is used to format a useful help text (which should be good enough for most of your purposes, especially if you don't want to reinvent the wheel).

  Note: the `helptext` attribute of your `BibFilter` subclass is only used by the default automatic filter option processor; so if you implement *format_help()* manually, the `helptext` attribute will be ignored.

CHAPTER 6

Python API: Core Bibolamazi Module

6.1 Module contents

Core bibolamazi module.

See *bibolamazi.core.bibfilter*, *bibolamazi.core.bibolamazifile*, *bibolamazi.core.bibusercache* for the main core modules.

6.2 Subpackages

6.2.1 bibolamazi.core.bibfilter package

bibolamazi.core.bibfilter.argtypes module

class bibolamazi.core.bibfilter.argtypes.**CommaStrList**(*iterable=[]*)
Bases: list

A list of values, specified as a comma-separated string.

class bibolamazi.core.bibfilter.argtypes.**CommaStrListArgType**

class bibolamazi.core.bibfilter.argtypes.**EnumArgType**(*listofvalues*)

bibolamazi.core.bibfilter.argtypes.**enum_class**(*class_name*, *values*, *default_value=0*, *value_attr_name='value'*)

class_name is the class name.

values should be a list of tuples *(string_key, numeric_value)* of all the expected string names and of their corresponding numeric values.

default_value should be the value that would be taken by default, e.g. by using the default constructor.

value_attr_name the name of the attribute in the class that should store the value. For example, the *arxiv* module defines the enum class *Mode* this way with the attribute *mode*, so that the numerical mode can be obtained with *enumobject.mode*.

bibolamazi.core.bibfilter.factory module

class bibolamazi.core.bibfilter.factory.**DefaultFilterOptions**(*filtername*, *fclass=None*)

filterDeclOptions()
This gives a list of *_ArgDoc* named tuples.

filterOptions()
This gives a list of *_ArgDoc* named tuples.

filterVarOptions()
This gives a list of *_ArgDoc* named tuples.

filtername()

format_filter_help()

getArgNameFromSOpt(*x*)

getSOptNameFromArg(*x*)

optionSpec(*argname*)

parse_optionstring(*optionstring*)
Parse the given option string (one raw string, which may contain quotes, escapes etc.) into arguments which can be directly provided to the filter constructor.

parse_optionstring_to_optspec(*optionstring*)
Parses the optionstring, and returns a description of which options where specified, which which values.

This doesn't go as far as *parse_optionstring()*, which returns pretty much exactly how to call the filter constructor. This function is meant for example for the GUI, who needs to parse what the user specified, and not necessarily how to construct the filter itself.

Return a dictionary:

```
{
  "_args": <additional *pargs positional arguments>
  "kwargs": <keyword arguments>
}
```

The value of `_args` is either *None*, or a list of additional positional arguments if the filter accepts **args* (and hence the option parser too). These will only be passed to **args* and NOT be distributed to the declared arguments of the filter constructor.

The value of `kwargs` is a dictionary of all options specified by keywords, either with the `--keyword=value` syntax or with the syntax `-sKey=Value`. The corresponding value is converted to the type the filter expects, in each case whenever possible (i.e. documented by the filter).

parser()

use_auto_case()

class bibolamazi.core.bibfilter.factory.**FilterArgumentParser**(*filtername*, ***kwargs*)
Bases: argparse.ArgumentParser

error(*message*)

exit(*status=0*, *message=None*)

exception bibolamazi.core.bibfilter.factory.**FilterCreateArgumentError**(*errorstr*, *name=None*)
Bases: *bibolamazi.core.bibfilter.factory.FilterError*

Although the filter arguments may have been successfully parsed, they may still not translate to a valid python filter call (i.e. in terms of function arguments, for example when using both positional and keyword arguments). This error is raised when the composed filter call is not valid.

fmt(*name*)

exception bibolamazi.core.bibfilter.factory.**FilterCreateError**(*errorstr*, *name=None*)
Bases: *bibolamazi.core.bibfilter.factory.FilterError*

There was an error instantiating the filter. This could be due because the filter constructor raised an exception.

fmt(*name*)

exception bibolamazi.core.bibfilter.factory.**FilterError**(*errorstr*, *name=None*)
Bases: exceptions.Exception

Signifies that there was some error in creating or instanciating the filter, or that the filter has a problem. (It could be, for example, that a function defined by the filter does not behave as expected. Or, that the option string passed to the filter could not be parsed.)

This is meant to signify a problem occuring in this factory, and not in the filter. The filter classes themselves should raise *bibfilter.BibFilterError* in the event of an error inside the filter.

fmt(*name*)

setName(*name*)

exception bibolamazi.core.bibfilter.factory.**FilterOptionsParseError**(*errorstr*, *name=None*)
Bases: *bibolamazi.core.bibfilter.factory.FilterError*

Raised when there was an error parsing the option string provided by the user.

fmt(*name*)

exception bibolamazi.core.bibfilter.factory.**FilterOptionsParseErrorHintSInstead**(*errorstr*, *name=None*)
Bases: *bibolamazi.core.bibfilter.factory.FilterOptionsParseError*

As FilterOptionsParseError, but hinting that maybe -sOption=Value was meant instead of -dOption=Value.

fmt(*name*)

exception bibolamazi.core.bibfilter.factory.**NoSuchFilter**(*fname*, *errorstr=None*)
Bases: exceptions.Exception

Signifies that the requested filter was not found. See also *get_module()*.

exception bibolamazi.core.bibfilter.factory.**NoSuchFilterPackage**(*fpname*, *errorstr='No such filter package'*, *fpdir=None*)
Bases: exceptions.Exception

Signifies that the requested filter package was not found. See also *get_module()*.

class bibolamazi.core.bibfilter.factory.**PrependOrderedDict**(**args*, ***kwargs*)
Bases: collections.OrderedDict

An ordered dict that stores the items in the order where the first item is the one that was added/modified last.

item_at(*idx*)

set_at(*idx*, *key*, *value*)

set_items(*items*)

bibolamazi.core.bibfilter.factory.**detect_filter_package_listings**()

bibolamazi.core.bibfilter.factory.**detect_filters**(*force_redetect=False*)

bibolamazi.core.bibfilter.factory.**filter_arg_parser**(*name*)
If the filter *name* uses the default-based argument parser, then returns a DefaultFilterOptions object that is initialized with the options available for the given filter *name*.

If the filter has its own option parsing mechanism, this returns *None*.

bibolamazi.core.bibfilter.factory.**filter_uses_default_arg_parser**(*name*)

bibolamazi.core.bibfilter.factory.**format_filter_help**(*filtname*)

bibolamazi.core.bibfilter.factory.**get_filter_class**(*name*, *filterpackage=None*)

bibolamazi.core.bibfilter.factory.**get_module**(*name*, *raise_nosuchfilter=True*, *filterpackage=None*)

bibolamazi.core.bibfilter.factory.**load_precompiled_filters**(*filterpackage*, *precompiled_modules*)

***filterpackage*: name of the filter package under which to scope the given precompiled** filter modules.

***precompiled_modules*: a dictionary of *'filter_name': filter_module* of precompiled** filter modules, along with their name.

bibolamazi.core.bibfilter.factory.**make_filter**(*name*, *optionstring*)

bibolamazi.core.bibfilter.factory.**reset_filters_cache**()

bibolamazi.core.bibfilter.factory.**validate_filter_package**(*fpname*, *fpdir*, *raise_exception=True*)

Module contents

class bibolamazi.core.bibfilter.**BibFilter**(**pargs*, ***kwargs*)
Bases: object

Base class for a *bibolamazi* filter.

To write new filters, you should subclass *BibFilter* and reimplement the relevant methods. See documentation of the different methods below to understand which to reimplement.

Constructor. No particular arguments are expected; any received are passed further to superclasses.

BIB_FILTER_BIBOLAMAZIFILE = 3
A constant that indicates that the filter should act upon the whole bibliography at once. See documentation for the *action()* method for more details.

BIB_FILTER_SINGLE_ENTRY = 1
A constant that indicates that the filter should act upon individual entries only. See documentation for the *action()* method for more details.

action()
Return one of *BIB_FILTER_SINGLE_ENTRY* or *BIB_FILTER_BIBOLAMAZIFILE*, which tells how this filter should function. Depending on the return value of this function, either *filter_bibentry()* or *filter_bibolamazifile()* will be called.

If the filter wishes to act on individual entries (like the built-in *arxiv* or *url* filters), then the subclass should return *BibFilter.BIB_FILTER_SINGLE_ENTRY*. At the time of filtering the data, the function *filter_bibentry()* will be called repeatedly for each entry of the database.

If the filter wishes to act on the full database at once (like the built-in *duplicates* filter), then the subclass should return *BIB_FILTER_BIBOLAMAZIFILE*. At the time of filtering the data, the function *filter_bibolamazifile()* will be called once with the full BibolamaziFile object as parameter. Note this is the only way to add or remove entries to or from the database, or to change their order.

Note that when the filter is instantiated by a `BibolamaziFile` (as is most of the time in practice), then the function *bibolamaziFile()* will always return a valid object, independently of the filter's way of acting.

bibolamaziFile()
Get the `BibolamaziFile` object that we are acting on. (The one previously set by *setBibolamaziFile()*.)

There's no use overriding this.

cacheAccessor(*klass*)
A shorthand for calling the *cacheAccessor()* method of the bibolamazi file returned by *bibolamaziFile()*.

filter_bibentry(*x*)
The main filter function for filters that filter the data entry by entry.

If the subclass' *action()* function returns *BibFilter.BIB_FILTER_SINGLE_ENTRY*, then the subclass must reimplement this function. Otherwise, this function is never called.

The object *x* is a `pybtex.database.Entry` object instance, which should be updated according to the filter's action and purpose.

The return value of this function is ignored. Subclasses should report warnings and logging through Python's logging mechanism (see doc of `core.blogger`) and should raise errors as *BibFilterError* (preferrably, a subclass). Other raised exceptions will be interpreted as internal errors and will open a debugger.

filter_bibolamazifile(*x*)
The main filter function for filters that filter the data entry by entry.

If the subclass' *action()* function returns *BibFilter.BIB_FILTER_SINGLE_ENTRY*, then the subclass must reimplement this function. Otherwise, this function is never called.

The object *x* is a `BibolamaziFile` object instance, which should be updated according to the filter's action and purpose.

The return value of this function is ignored. Subclasses should report warnings and logging through Python's logging mechanism (see doc of `core.blogger`) and should raise errors as *BibFilterError* (preferrably, a subclass). Other raised exceptions will be interpreted as internal errors and will open a debugger.

classmethod getHelpAuthor()
Convenience function that returns *helpauthor*, with whitespace stripped. Use this when getting the contents of the helpauthor text.

There's no need to (translate: you should not) reimplement this function in your subclass.

classmethod getHelpDescription()
Convenience function that returns *helpdescription*, with whitespace stripped. Use this when getting the contents of the helpdescription text.

There's no need to (translate: you should not) reimplement this function in your subclass.

classmethod getHelpText()
Convenience function that returns *helptext*, with whitespace stripped. Use this when getting the contents of the helptext text.

There's no need to (translate: you should not) reimplement this function in your subclass.

getRunningMessage()
Return a nice message to display when invoking the fitler. The default implementation returns *name()*.

Define this to whatever you want in your subclass to describe what you're doing. The core bibolamazi program displays this information to the user as it runs the filter.

helpauthor = ''
Your subclass should provide a *helpauthor* attribute, containing a one-line notice with the name of the author that wrote the filter code. You may also add a copyright notice. The exact format is not specified. This text is typically displayed at the top of the page generated by `bibolamazi --help <filter>`.

You should also avoid accessing this class attribute, you should use *getHelpAuthor()* instead, which will ensure that whitespace is properly stripped.

helpdescription = 'Some filter that filters some entries'
Your subclass should provide a *helpdescription* attribute, containing a one-line description of what your filter does. This is typically displayed when invoking `bibolamazi --list-filters`, along with the filter name.

You should also avoid accessing this class attribute, you should use *getHelpDescription()* instead, which will ensure that whitespace is properly stripped.

helptext = ''
Your subclass should provide a *helptext* attribute, containing a possibly long, as detailed as possible description of how to use your filter. You don't need to provide the basic 'usage' and option list, which are automatically generated; but you should include all the text that would appear after the option list. This is typically displayed when invoking `bibolamazi --help <filter>`.

You should also avoid accessing this class attribute, you should use *getHelpText()* instead, which will ensure that whitespace is properly stripped.

name()
Returns the name of the filter as it was invoked in the bibolamazifile. This might be with, or without, the filterpackage. This information should be only used for reporting purposes and might slightly vary.

If the filter was instantiated manually, and *setInvokationName()* was not called, then this function returns the class name.

The subclass should not reimplement this function unless it really really really *really* feels it needs to.

prerun(*bibolamazifile*)
This function gets called immediately before the filter is run, after any preceeding filters have been executed.

It is not very useful if the *action()* is *BibFilter.BIB_FILTER_BIBOLAMAZIFILE*, but it can prove useful for filters with action *BibFilter.BIB_FILTER_SINGLE_ENTRY*, if any sort of preprocessing task should be done just before the actual filtering of the data.

The default implementation does nothing.

requested_cache_accessors()
This function should return a list of `bibusercache.BibUserCacheAccessor` class names of cache objects it would like to use. The relevant caches are then collected from the various filters and automatically instantiated and initialized.

The default implementation of this function returns an empty list. Subclasses should override if they want to access the bibolamazi cache.

setBibolamaziFile(*bibolamazifile*)
Remembers *bibolamazifile* as the `BibolamaziFile` object that we will be acting on.

There's no use overriding this. When writing filters, there's also no need calling this explicitly, it's done in `BibolamaziFile`.

setInvokationName (*filtername*)
Called internally by bibolamazifile, so that *name()* returns the name by which this filter was invoked.

This function sets exactly what *name()* will return. Subclasses should not reimplement, the default implementation should suffice.

exception bibolamazi.core.bibfilter.**BibFilterError** (*filtername*, *message*)
Bases: *bibolamazi.core.butils.BibolamaziError*

Exception a filter should raise if it encounters an error.

6.2.2 bibolamazi.core.bibusercache package

bibolamazi.core.bibusercache.tokencheckers module

This module provides a collection of useful token checkers that can be used to make sure the cache information is always valid and up-to-date.

Recall the Bibolamazi Cache is organized as nested dictionaries in which the cached information is organized.

One main concern of the caching mechanism is that information be *invalidated* when it is no longer relevant (between different runs of bibolamazi). This may be for example because the original bibtex entry from the source has changed.

Each cache dictionary (`BibUserCacheDic`) may be set a *token validator*, that is a verifier instance class which will invalidate items it detects as no longer valid. The validity of items is determined on the basis of *validation tokens*.

When an item in a cache dictionary is added or updated, a token (which can be any python value) is generated corresponding to the cached value. This token may be, for example, the date and time at which the value was cached. The validator then checks the tokens of the cache values and detects those entries whose token indicates that the entries are no longer valid: for example, if the token corresponds to the date and time at which the entry was stored, the validator may invalidate all entries whose token indicates that they are too old.

Token Checkers are free to decide what information to store in the tokens. See the `tokencheckers` module for examples. Token checkers must derive from the base class `TokenChecker`.

class bibolamazi.core.bibusercache.tokencheckers.**EntryFieldsTokenChecker** (*bibdata*, *fields=[]*, *store_type=False*, *store_persons=[]*, ***kwargs*)
Bases: *bibolamazi.core.bibusercache.tokencheckers.TokenChecker*

A *TokenChecker* implementation that checks whether some fields of a bibliography entry have changed.

This works by calculating a MD5 hash of the contents of the given fields.

Constructs a token checker that will invalidate an entry if any of its fields given here have changed.

bibdata is a reference to the bibolamazifile's bibliography data; this is the return value of `bibolamaziData()`.

fields is a list of bibtex fields which should be checked for changes. Note that the 'author' and 'editor' fields are treated specially, with the *store_persons* argument.

If *store_type* is *True*, the entry is also invalidated if its type changes (for example, from '@unpublished' to '@article').

store_persons is a list of person roles we should check for changes (see person roles in `pybtex.database.Entry` : this is either 'author' or 'editor'). Specify for example 'author' here instead of in the *fields* argument. This is because *pybtex* treats the 'author' and 'editor' fields specially.

new_token (*key*, *value*, ***kwargs*)

class bibolamazi.core.bibusercache.tokencheckers.**TokenChecker**(***kwargs*)

Bases: object

Base class for a token checker validator.

The *new_token()* function always returns *True* and *cmp_tokens()* just compares tokens for equality with the == operator.

Subclasses should reimplement *new_token()* to return something useful. Subclasses may either use the default implementation equality comparision for *cmp_tokens()* or reimplement that function for custom token validation condition (e.g. as in *TokenCheckerDate*).

cmp_tokens(*key*, *value*, *oldtoken*, ***kwargs*)

Checks to see if the dictionary entry *(key, value)* is still up-to-date and valid. The old token, returned by a previous call to *new_token()*, is provided in the argument *oldtoken*.

The default implementation calls *new_token()* for the *(key, value)* pair and compares the new token with the old token *oldtoken* for equality with the == operator. Depending on your use case, this may be enough so you may not have to reimplement this function (as, for example, in *EntryFieldsTokenChecker*).

However, you may wish to reimplement this function if a different comparision method is required. For example, if the token is a date at which the information was retrieved, you might want to test how old the information is, and invalidate it only after it has passed a certain amount of time (as done in *TokenCheckerDate*).

It is advisable that code in this function should be protected against having the wrong type in *oldtoken* or being given *None*. Such cases might easily pop up say between Bibolamazi Versions, or if the cache was once not properly set up. In any case, it's safer to trap exceptions here and return *False* to avoid an exception propagating up and causing the whole cache load process to fail.

Return *True* if the entry is still valid, or *False* if the entry is out of date and should be discarded.

new_token(*key*, *value*, ***kwargs*)

Return a token which will serve to identify changes of the dictionary entry *(key, value)*. This token may be any Python picklable object. It can be anything that *cmp_tokens()* will undertsand.

The default implementation returns *True* all the time. Subclasses should reimplement to do something useful.

class bibolamazi.core.bibusercache.tokencheckers.**TokenCheckerCombine**(**args*, ***kwargs*)

Bases: *bibolamazi.core.bibusercache.tokencheckers.TokenChecker*

A *TokenChecker* implementation that combines several different token checkers. A cache entry is deemed valid only if it considered valid by all the installed token checkers.

For example, you may want to both make sure the cache has the right version (with a *VersionTokenChecker* and that it is up-to-date).

Constructor. Pass as arguments here instances of token checkers to check for, e.g.:

```
chk = TokenCheckerCombine(
    VersionTokenChecker('2.0'),
    EntryFieldsTokenChecker(bibdata, ['title', 'journal'])
    )
```

cmp_tokens(*key*, *value*, *oldtoken*, ***kwargs*)

new_token(*key*, *value*, ***kwargs*)

class bibolamazi.core.bibusercache.tokencheckers.**TokenCheckerDate**(*time_valid=datetime.timedelta(5)*, ***kwargs*)

Bases: *bibolamazi.core.bibusercache.tokencheckers.TokenChecker*

A *TokenChecker* implementation that remembers the date and time at which an entry was set, and invalidates the entry after an amount of time *time_valid* has passed.

The amount of time the information remains valid is given in the *time_valid* argument of the constructor or is set with a call to *set_time_valid()*. In either case, you should provide a python `datetime.time_delta` object.

cmp_tokens(*key*, *value*, *oldtoken*, ***kwargs*)

new_token(***kwargs*)

set_time_valid(*time_valid*)

class bibolamazi.core.bibusercache.tokencheckers.**TokenCheckerPerEntry**(*checkers={}*, ***kwargs*)

Bases: *bibolamazi.core.bibusercache.tokencheckers.TokenChecker*

A *TokenChecker* implementation that associates different *TokenChecker*'s for individual entries, set manually.

By default, the items of the dictionary are always valid. When an entry-specific token checker is set with *add_entry_check()*, that token checker is used for that entry only.

add_entry_check(*key*, *checker*)
Add an entry-specific checker.

key is the entry key for which this token checker applies. *checker* is the token checker instance itself. It is possible to make several keys share the same token checker instance.

Note that no explicit validation is performed. (This can't be done because we don't even have a pointer to the cache dict.) So you should call manually `BibUserCacheDict.validate_item()`

If a token checker was already set for this entry, it is replaced by the new one.

checker_for(*key*)
Returns the token instance that has been set for the entry *key*, or *None* if no token checker has been set for that entry.

cmp_tokens(*key*, *value*, *oldtoken*, ***kwargs*)

has_entry_for(*key*)
Returns *True* if we have a token checker set for the given entry *key*.

new_token(*key*, *value*, ***kwargs*)

remove_entry_check(*key*)
As the name suggests, remove the token checker associated with the given entry key *key*. If no token checker was previously set, then this function does nothing.

class bibolamazi.core.bibusercache.tokencheckers.**VersionTokenChecker**(*this_version*, ***kwargs*)

Bases: *bibolamazi.core.bibusercache.tokencheckers.TokenChecker*

A *TokenChecker* which checks entries with a given version number.

This is useful if you might change the format in which you store entries in your cache: adding a version number will ensure that any old-formatted entries will be discarded.

Constructs a version validator token checker.

this_version is the current version. Any entry that was not exactly marked with the version *this_version* will be deemed invalid.

this_version may actually be any python object. Comparision is done with the equality operator == (actually using the original `TokenChecker` implementation).

new_token(*key*, *value*, ***kwargs*)

Module contents

class bibolamazi.core.bibusercache.**BibUserCache**(*cache_version=None*)
Bases: object

The basic root cache object.

This object stores the corresponding cache dictionaries for each cache. (See `cacheFor()`.)

(Internally, the caches are stored in one root `BibUserCacheDic`.)

cacheExpirationTokenChecker()
Returns a cache expiration token checker validator which is configured with the default cache invalidation time.

This object may be used by subclasses as a token checker for sub-caches that need regular invalidation (typically several days in the default configuration).

Consider using though *installCacheExpirationChecker()*, which simply applies a general validator to your full cache; this is generally what you might want.

cacheFor(*cache_name*)
Returns the cache dictionary object for the given cache name. If the cache dictionary does not exist, it is created.

hasCache()
Returns *True* if we have any cache at all. This only returns *False* if there are no cache dictionaries defined.

installCacheExpirationChecker(*cache_name*)
Installs a cache expiration checker on the given cache.

This is a utility that is at the disposal of the cache accessors to easily set up an expiration validator on their caches. Also, a single instance of an expiry token checker (see *TokenCheckerDate*) is shared between the different sub-caches and handled by this main cache object.

The duration of the expiry is typically several days; because the token checker instance is shared this cannot be changed easily nor should it be relied upon. If you have custom needs or need more control over this, create your own token checker.

Returns: the cache dictionary. This may have changed to a new empty object if the cache didn't validate!

WARNING: the cache dictionary may have been altered with the validation of the cache! Use the return value of this function, or call `BibUserCacheAccessor.cacheDic()` again!

Note: this validation will not validate individual items in the cache dictionary, but the dictionary as a whole. Depending on your use case, it might be worth introducing per-entry validation. For that, check out the various token checkers in `tokencheckers` and call `set_validation()` to install a specific validator instance.

loadCache(*cachefobj*)
Load the cache from a file-like object *cachefobj*.

This tries to unpickle the data and restore the cache. If the loading fails, e.g. because of an I/O error, the exception is logged but ignored, and an empty cache is initialized.

Note that at this stage only the basic validation is performed; the cache accessors should then each initialize their own subcaches with possibly their own specialized validators.

saveCache(*cachefobj*)
Saves the cache to the file-like object *cachefobj*. This dumps a pickle-d version of the cache information into the stream.

setDefaultInvalidationTime(*time_delta*)
A timedelta object giving the amount of time for which data in cache is consdered valid (by default).

class bibolamazi.core.bibusercache.**BibUserCacheAccessor**(*cache_name*, *bibolamazifile*, ***kwargs*)

Bases: object

Base class for a cache accessor.

Filters should access the bibolamazi cache through a *cache accessor*. A cache accessor organizes how the caches are used and maintained. This is needed since several filters may want to access the same cache (e.g. fetched arXiv info from the arxiv.org API), so it is necessary to abstract out the cache object and how it is maintained out of the filter. This also avoids issues such as which filter is responsible for creating/refreshing the cache, etc.

A unique accessor instance is attached to a particular cache name (e.g. 'arxiv_info'). It is instantiated by the BibolamaziFile. It is instructed to initialize the cache, possibly install token checkers, etc. at the beginning, before running any filters. The accessor is free to handle the cache as it prefers–build it right away, refresh it on demand only, etc.

Filters access the cache by requesting an instance to the accessor. This is done by calling `cache_accessor()` (you can use `bibolamaziFile()` to get a pointer to the *bibolamazifile* object.). Filters should declare in advance which caches they would like to have access to by reimplementing the `requested_cache_accessors()` method.

Accessors are free to implement their public API how they deem it best. There is no obligation or particular structure to follow. (Although *refresh_cache()*, *fetch_missing_items(list)*, or similar function names may be typical.)

Cache accessor objects are instantiated by the bibolamazi file. Their constructors should accept a keyword argument *bibolamazifile* and pass it on to the superclass constructor. Constructors should also accept ***kwargs* for possible compatibility with future additions and pass it on to the parent constructor. The *cache_name* argument of this constructor should be a fixed string passed by the subclass, identifying this cache (e.g. 'arxiv_info').

bibolamaziFile()
Returns the parent bibolamazifile of this cache accessor. This may be useful, e.g. to initialize a token cache validator in *initialize()*.

Returns the object given in the constructor argument. Do not reimplement this function.

cacheDic()
Returns the cache dictionary. This is meant as a 'protected' method for the accessor only. Objects that query the accessor should use the accessor-specific API to access data.

The cache dictionary is a *BibUserCacheDic* object. In particular, subcaches may want to set custom token checkers for proper cache invalidation (this should be done in the *initialize()* method).

This returns the data in the cache object that was set internally by the `BibolamaziFile` via the method *setCacheObj()*. Don't call that manually, though, unless you're implementing an alternative `BibolamaziFile` class !

cacheName()
Return the cache name, as set in the constructor.

Subclasses do not need to reimplement this function.

cacheObject()
Returns the parent *BibUserCache* object in which *cacheDic()* is a sub-cache. This is provided FOR CONVENIENCE! Don't abuse this!

You should never need to access the object directly. Maybe just read-only to get some standard attributes such as the root cache version. If you're writing directly to the root cache object, there is most likely a design flaw in your code!

Most of all, don't write into other sub-caches!!

initialize(*cache_obj*)
Initialize the cache.

Subclasses should perform any initialization tasks, such as install *token checkers*. This function should not return anything.

Note that it is *strongly* recommended to install some form of cache invalidation, would it be just even an expiry validator. You may want to call `installCacheExpirationChecker()` on *cache_obj*.

Note that the order in which the *initialize()* method of the various caches is called is undefined.

Use the *cacheDic()* method to access the cache dictionary. Note that if you install token checkers on this cache, e.g. with *cache_obj.installCacheExpirationChecker()*, then the cache dictionary object may have changed! (To be sure, call *cacheDic()* again.)

The default implementation raises a *NotImplemented* exception.

setCacheObj(*cache_obj*)
Sets the cache dictionary and cache object that will be returned by *cacheDic()* and *cacheObject()*, respectively. Accessors and filters should not call (nor reimplement) this function. This function gets called by the *BibolamaziFile*.

class `bibolamazi.core.bibusercache.`**BibUserCacheDic**(**args*, ***kwargs*)
Bases: `_abcoll.MutableMapping`

Implements a cache where information may be stored between different runs of bibolamazi, and between different filter runs.

This is a dictionary of key=value pairs, and can be used like a regular python dictionary.

This implements *cache validation*, i.e. making sure that the values stored in the cache are up-to-date. Each entry of the dictionary has a corresponding *token*, i.e. a value (of any python picklable type) which will identify whether the cache is invalid or not. For example, the value could be *datetime* corresponding to the time when the entry was created, and the rule for validating the cache might be to check that the entry is not more than e.g. 3 days old.

child_notify_changed(*obj*)

iteritems()

new_value_set(*key=None*)
Informs the dic that the value for *key* has been updated, and a new validation token should be stored.

If *key* is *None*, then this call is meant for the current object, so this call will relay to the parent dictionary.

set_parent(*parent*)

set_validation(*tokenchecker*, *validate=True*)
Set a function that will calculate the *token' for a given entry, for cache validation. The function 'fn* shall compute a value based on a key (and possibly cache value) of the cache, such that comparision with *fncmp* (by default equality) will tell us if the entry is out of date. See the documentation for the `tokencheckers` modules for more information about cache validation.

If *validate* is *True*, then we immediately validate the contents of the cache.

validate()
Validate this whole dictionary, i.e. make sure that each entry is still valid.

This calls *validate_item()* for each item in the dictionary.

validate_item(*key*)
Validate an entry of the dictionary manually. Usually not needed.

If the value is valid, and happens to be a BibUserCacheDic, then that dictionary is also validated.

Invalid entries are deleted.

Returns *True* if have valid item, otherwise *False*.

exception bibolamazi.core.bibusercache.**BibUserCacheError**(*cache_name*, *message*)
Bases: *bibolamazi.core.butils.BibolamaziError*

An exception which occurred when handling user caches. Usually, problems in the cache are silently ignored, because the cache can usually be safely regenerated.

However, if there is a serious error which prevents the cache from being regenerated, for example, then this error should be raised.

class bibolamazi.core.bibusercache.**BibUserCacheList**(**args*, ***kwargs*)
Bases: _abcoll.MutableSequence

append(*value*)

insert(*index*, *value*)

6.3 bibolamazi.core.argparseactions module

This module defines callbacks and actions for parsing the command-line arguments for bibolamazi. You're most probably not interested in this API. (Not mentioning that it might change if I feel the need for it.)

bibolamazi.core.argparseactions.**help_list_filters**()

bibolamazi.core.argparseactions.**helptext_prolog**()

class bibolamazi.core.argparseactions.**opt_action_help**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class bibolamazi.core.argparseactions.**opt_action_version**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class bibolamazi.core.argparseactions.**opt_init_empty_template**(*nargs=1*, ***kwargs*)
Bases: argparse.Action

class bibolamazi.core.argparseactions.**opt_list_filters**(*nargs=0*, ***kwargs*)
Bases: argparse.Action

class bibolamazi.core.argparseactions.**opt_set_fine_log_levels**(*nargs=1*, ***kwargs*)
Bases: argparse.Action

class bibolamazi.core.argparseactions.**opt_set_verbosity**(*nargs=1*, ***kwargs*)
Bases: argparse.Action

bibolamazi.core.argparseactions.**run_pager**(*text*)
Call *pydoc.pager()* in a unicode-safe way.

class bibolamazi.core.argparseactions.**store_key_bool**(*option_strings*, *dest*, *nargs=1*, *const=True*, *exception=<type 'exceptions.ValueError'>*, ***kwargs*)
Bases: argparse.Action

Handles an ghostscript-style option of the type '-dBoolKey' or '-dBoolKey=0'.

class bibolamazi.core.argparseactions.**store_key_const**(*option_strings*, *dest*, *nargs=1*, *const=True*, ***kwargs*)
Bases: argparse.Action

class bibolamazi.core.argparseactions.**store_key_val**(*option_strings*, *dest*, *nargs=1*, *exception=<type 'exceptions.ValueError'>*, ***kwargs*)
Bases: argparse.Action

Handles an ghostscript-style option of the type '-sBoolKey=some-value'.

class bibolamazi.core.argparseactions.**store_or_count**(*option_strings*, *dest*, *nargs='?'*, ***kwargs*)
Bases: argparse.Action

6.4 bibolamazi.core.bibolamazifile module

The Main bibolamazifile module: this contains the *BibolamaziFile* class definition.

bibolamazi.core.bibolamazifile.**AFTER_CONFIG_TEXT = "%\n%\n% ALL CHANGES BEYOND THIS POINT W**
Some text which is inserted immediately after the config section when saving bibolamazi files. Includes a warning about losing any changes.

bibolamazi.core.bibolamazifile.**BIBOLAMAZIFILE_INIT = 0**
Bibolamazi file load state: freshly initialized, no data read. See doc for *BibolamaziFile*.

bibolamazi.core.bibolamazifile.**BIBOLAMAZIFILE_LOADED = 3**
Bibolamazi file load state: data read and parsed, filters instanciated and data from sources loaded. See doc for *BibolamaziFile*.

bibolamazi.core.bibolamazifile.**BIBOLAMAZIFILE_PARSED = 2**
Bibolamazi file load state: data read and parsed, filters instanciated but no sources loaded. See doc for *BibolamaziFile*.

bibolamazi.core.bibolamazifile.**BIBOLAMAZIFILE_READ = 1**
Bibolamazi file load state: data read, not parsed. See doc for *BibolamaziFile*.

bibolamazi.core.bibolamazifile.**BIBOLAMAZI_FILE_ENCODING = 'utf-8'**
The encoding used to read and write bibolamazi files. Don't change this.

class bibolamazi.core.bibolamazifile.**BibolamaziFile**(*fname=None*, *create=False*, *load_to_state=3*, *use_cache=True*, *default_cache_invalidation_time=None*)
Bases: object

Represents a Bibolamazi file.

This class provides an API to read and parse bibolamazi files, as well as load data defined in its configuration section and interact with its filters.

A *BibolamaziFile* object may be in different load states:

- *BIBOLAMAZIFILE_INIT*: The *BibolamaziFile* object is initialized to an empty state. The file name (*fname()*) may be set already, but is *None* by default.
- *BIBOLAMAZIFILE_READ*: Data has been read from a given file, but not parsed. You may call certain methods such as *rawHeader()* or *configData()*, but e.g. *configCmds()* will return an invalid value.
- *BIBOLAMAZIFILE_PARSED*: Data has been read from a bibolamazi file and parsed, and filter objects have been instanciated. Methods such as *filters()* or *sourceLists()* may be called.
- *BIBOLAMAZIFILE_LOADED*: The bibolamazi file has been read and parsed, filter objects have been instanciated and bibtex data from the sources has been loaded. This is the "maximally loaded" state.

You may query the load state with *getLoadState()* and load a bibolamazi file either from the constructor or by calling explicitly *load()*. Some methods on this object may only be called if the object has reached a certain load state. These methods are documented as such.

The bibliography database is accessed with *bibliographyData()*. You may change the entries in the database via direct access (using the *pybtex* API), or using the method *setEntries()*.

To create a new bibolamazi file template, you may specify *create=True* to the constructor with a valid file name, and save the object.

Create a BibolamaziFile object.

If *fname* is provided, the file is fully loaded (unless *create* is also provided).

If *create* is given and set to *True*, then an empty template is loaded and the internal file name is set to *fname*. The internal state will be set to *BIBOLAMAZIFILE_LOADED* and calling *saveToFile()* will result in writing this template to the file *fname*.

If *load_to_state* is given, then the file is only loaded up to the given state. See *load()* for more details. The state should be one of *BIBOLAMAZIFILE_INIT*, *BIBOLAMAZIFILE_READ*, *BIBOLAMAZIFILE_PARSED* or *BIBOLAMAZIFILE_LOADED*.

If *use_cache* is *True* (default), then when loading this file, we'll attempt to load a corresponding cache file if it exists. Note that even if *use_cache* is *False*, then cache will still be *written* when calling *saveToFile()*.

If *default_cache_invalidation_time* is given, then the default cache invalidation time is set before loading the cache.

bibliographyData()
Return the *pybtex.database.BibliographyData* object which stores all the bibliography entries.

This object is only instanciated and initialized once in the *BIBOLAMAZIFILE_LOADED* state. If `getLoadState() != BIBOLAMAZIFILE_LOADED`, then this function returns *None*.

bibliographydata()
Deprecated since version 2.0: Use *bibliographyData()* instead!

cacheAccessor(*klass*)
Returns the cache accessor instance corresponding to the given class.

See documentation of *bibolamazi.core.bibusercache* for more information about the bibolamazi cache.

cacheFileName()
The file name where the cache will be stored. You don't need to access this directly, the cache will be loaded and saved automatically.

Filters should only access the cache through cache accessors. See *cacheAccessor()*.

configCmds()
Return a list of parsed commands from the configuration section.

This returns a list of *BibolamaziFileCmd* objects.

This may be called in the state *BIBOLAMAZIFILE_PARSED*.

configData()
Returns the configuration commands, with leading percent signs stripped, and without the begin and end tags.

This may be called in the state *BIBOLAMAZIFILE_READ*.

configLineNo(*filelineno*)
Utility to convert file line number to config line number

Returns the line number in the config data corresponding to line *filelineno* in the file. Opposite of *fileLineNo()*.

This may be called in the state *BIBOLAMAZIFILE_READ*.

fdir()
Returns the directory name in which this bibolamazi file resides, always as a full path (using *os.path.realpath*, resolving symlinks). The value is cached, so you may call this function several times with little performance overhead.

If *fname()* is *None* (this is only possible if the load state is *BIBOLAMAZIFILE_INIT*), then *None* is returned.

fileLineNo(*configlineno*)
Utility to convert config line number to file line number

Returns the line number in the bibolamazi file corresponding to the config line number *configlineno*. The *configlineno* refers to the line number INSIDE the config section, where line number 1 is right after the begin config tag *CONFIG_BEGIN_TAG*.

This may be called in the state *BIBOLAMAZIFILE_READ*.

filters()
Return a list of filter instances

This returns the list of all filter commands given in the bibolamazi config section. The instances have already been instanciated with the proper options. The order of this list is exactly the order of the filters in the config section.

If in the config section the same filter is invoked several times, then separate instances are returned in this list with the appropriate ordering, as you'd expect.

fname()
Returns the file name this object refers to.

If the state is any other than *BIBOLAMAZIFILE_INIT*, then this function will never return *None*.

getLoadState()
Returns the state of the *BibolamaziFile* object. One of *BIBOLAMAZIFILE_INIT*, *BIBOLAMAZIFILE_READ*, *BIBOLAMAZIFILE_PARSED*, or *BIBOLAMAZIFILE_LOADED*.

load(*fname=[]*, *to_state=3*)
Load the given file into the current object.

If *fname* is *None*, then reset the object to an empty state. If *fname* is not given or an empty `list`, then use any previously loaded fname and its state.

This function may be called several times with different states to incrementally load the file, for example:

```
bibolamazifile.reset()
# load up to 'parsed' state
bibolamazifile.load(fname="somefile.bibolamazi.bib", to_state=BIBOLAMAZIFILE_PARSED)
# continue loading up to fully 'loaded' state
bibolamazifile.load(fname="somefile.bibolamazi.bib", to_state=BIBOLAMAZIFILE_LOADED)
```

If *to_state* is given, will only attempt to load the file up to that state. This can be useful, e.g., in a config editor which needs to parse the sections of the file but does not need to worry about syntax errors. The state should be one of *BIBOLAMAZIFILE_INIT*, *BIBOLAMAZIFILE_READ*, *BIBOLAMAZIFILE_PARSED* or *BIBOLAMAZIFILE_LOADED*.

rawConfig()
Return the raw configuration section. The returned value will NOT have the leading percent signs removed.

This may be called in the state *BIBOLAMAZIFILE_READ*.

rawHeader()
Return any content above the configuration section.

This may be called in the state *BIBOLAMAZIFILE_READ*.

rawRest()
Return all the contents after the config section at the moment the file was read from the disk. This includes the begin and end config section tags (*CONFIG_BEGIN_TAG* and *CONFIG_END_TAG*).

Any changes to the bibliography data will not be reflected here, even if you call *saveToFile()*.

This may be called in the state *BIBOLAMAZIFILE_READ*.

rawStartConfigDataLineNo()
Returns the line number on which the begin config tag *CONFIG_BEGIN_TAG* is located. Line numbers start at 1 at the top of the file like in any reasonable editor.

This may be called in the state *BIBOLAMAZIFILE_READ*.

reset()
Reset the current object to an empty state and unset the file name. This will reset the object to the state *BIBOLAMAZIFILE_INIT*.

resolveSourcePath(*path*)
Resolves a path (for example corresponding to a source file) to an absolute file location.

This function expands '~/foo/bar' to e.g. '/home/someone/foo/bar'; it also expands shell variables, e.g. '$HOME/foo/bar' or '${MYBIBDIR}/foo/bar.bib'.

If the path is relative, it is made absolute by interpreting it as relative to the location of this bibolamazi file (see *fdir()*).

Note: *path* should not be an URL.

saveToFile()
Save the current bibolamazi file object to disk.

This will write to the file *fname()* in order:

- the raw header data (*rawHeader()*) unchanged
- the config section text (*rawConfig()*) unchanged
- the bibliography data contained in *bibliographyData()*, saved in BibTeX format.

A warning message is included after the config section that the remainder of the file was automatically generated.

As the file *fname* is expected to already exist, it is always silently overwritten (so be careful).

setBibliographyData(*bibliographydata*)
Set the *bibliographydata* database object directly.

The object *bibliographydata* should be of instance `pybtex.database.BibliographyData`.

> **Warning:** Filters should NOT set a different bibliographydata object: caches might have kept a pointer to this object (see, for example `EntryFieldsTokenChecker`). Please use *setEntries()* instead.

setConfigData(*configdata*)
Store the given data *configdata* in memory as the configuration section of this file.

This function cleanifies the *configdata* a bit by adding leading percent signs and forcing a final newline, adds the config section begin and end tags, and then directly calls *setRawConfig()*.

setDefaultCacheInvalidationTime(*time_delta*)
A timedelta object giving the amount of time for which data in cache is consdered valid (by default).

Note that this function should be called BEFORE the data is loaded. If you just call, for example the default constructor, this might be too late already. If you use the *load()* function, set the default cache invalidation time before you load up to the state *BIBOLAMAZIFILE_LOADED*.

Note that you may also use the option in the constructor *default_cache_invalidation_time*, which has the same effect as this funtion called at the appropriate time.

setEntries(*bibentries*)
Replace all the entries in the current bibliographydata object by the given entries.

Arguments:

- *bibentries*: the new entries to set. *bibentries* should be an iterable of *(key, entry)* (or, more precisely, any valid argument for `pybtex.database.BibliographyData.add_entries()`).

> **Warning:** This will remove any existing entries in the database.

This function alters the current *bibliographyData()* object, and does not replace it by a new object. (I.e., if you kept a reference to the *bibliographyData()* object, the reference is still valid after calling this function.)

setRawConfig(*configblock*)
Store the given *configblock* in memory as the raw configuration section of the bibolamazi file. We must be at least in state *BIBOLAMAZIFILE_READ* to call this function; this function will also reset to state back to *BIBOLAMAZIFILE_READ* (as the configuration might have changed).

Note that *configblock* is expected to start and end with the appropriate config section tags (*CONFIG_BEGIN_TAG* and *CONFIG_END_TAG*).

After calling this function, *configData()* will return the new configuration data. Call *load()* to re-instanciate filters and re-load sources.

sourceLists()
Return a list of source lists, in the order they are specified in the configuration section.

Each item in the returned list is itself a list of alternative sources to consider.

This may be called in the state *BIBOLAMAZIFILE_PARSED*.

sources()
Return a list of sources which have been read.

This is a list of strings. Each item in the returned list is one of the items in the corresponding list from *sourceLists()* (the one that was actually found and read). If no corresponding item in *sourceLists()* was readable, then the corresponding item in this list is *None*. For example:

```
# suppose that we have the following instructions in the bibolamazi file:
#
#     src: src1.bib
#     src: a.bib b.bib c.bib
#     src: x/x.bib y/y.bib
#
# we would then have:
#
f.sourceLists() == [["src1.bib"], ["a.bib", "b.bib", "c.bib"], ["x/x.bib", "y/y.bib"]]

# suppose that "src1.bib" exists, "a.bib" doesn't exist but "b.bib" exists, and neither
# "x/x.bib" nor "y/y.bib" don't exist.
#
# Then, after loading this object, we get:
#
f.sources() == ["src1.bib", "b.bib", None]
```

This function may be called in the state *BIBOLAMAZIFILE_LOADED*.

class bibolamazi.core.bibolamazifile.**BibolamaziFileCmd**(*cmd=None*, *text=''*, *lineno=-1*, *linenoend=-1*, *info={}*)

A command in the bibolamazi file configuration

Stores the command name (e.g. 'src' or 'filter'), additional text (the options), line number information and possible additional information.

Object Properties:

- *cmd*: the command name. Currently this is 'src' or 'filter'
- *text*: the text following the command. This is e.g. the sources list, or a filter name followed by options. In general, it is anything following the 'src:' or 'filter:' commands.
- *lineno*: the line number at which this command is specified in the bibolamazi file, relative to the top of the file. The first line of the file is line number 1.
- *linenoend*: the line number at which the command ends.
- *info*: a dictionary with possible additional information which is available at parse time. For example, the filter name for 'filter' commands is stored when parsing commands.

See also bibolamazifile.configCmds().

Construct a *BibolamaziFileCmd* with the given *cmd*, *text*, *lineno*, *linenoend* and *info*.

exception bibolamazi.core.bibolamazifile.**BibolamaziFileParseError**(*msg*, *fname=None*, *lineno=None*)

Bases: *bibolamazi.core.butils.BibolamaziError*

bibolamazi.core.bibolamazifile.**CONFIG_BEGIN_TAG = '%%%-BIB-OLA-MAZI-BEGIN-%%%'**

The line which defines the beginning of a config section in a bibolamazi file.

bibolamazi.core.bibolamazifile.**CONFIG_END_TAG = '%%%-BIB-OLA-MAZI-END-%%%'**

The line which defines the end of a config section in a bibolamazi file.

exception bibolamazi.core.bibolamazifile.**NotBibolamaziFileError**(*msg*, *fname=None*, *lineno=None*)

Bases: *bibolamazi.core.bibolamazifile.BibolamaziFileParseError*

This error is raised to signify that the file specified is not a bibolamazi file—most probably, it does not contain a valid configuration section.

6.5 bibolamazi.core.blogger module

Set up a logging framework for logging debug, information, warning and error messages.

Modules should get their logger using Python's standard `logging` mechanism:

```
import logging
logger = logging.getLogger(__name__)
```

This allows for the user to be rather specific about which type of messages she/he would like to see.

class bibolamazi.core.blogger.**BibolamaziConsoleFormatter**(*ttycolors=False*, *show_pos_info_level=None*, ***kwargs*)

Bases: logging.Formatter

Format log messages for console output. Customized for bibolamazi.

format(*record*)

setShowPosInfoLevel(*level*)

class bibolamazi.core.blogger.**BibolamaziLogger**(*name*, *level=0*)

Bases: logging.Logger

A Logger used in Bibolamazi.

This logger class knows about an additional log level, LONGDEBUG.

Initialize the logger with a name and an optional level.

getSelfLevel()

Returns the level that was set on this logger. If no specific level was set, then returns *logging.NOTSET*. In this respect, this is NOT the same as *getEffectiveLevel()*.

longdebug(*msg*, **args*, ***kwargs*)

Produce a log message at level LONGDEBUG.

class bibolamazi.core.blogger.**ConditionalFormatter**(*defaultfmt=None*, *datefmt=None*, ***kwargs*)

Bases: logging.Formatter

A formatter class.

Very much like logging.Formatter, except that different formats can be specified for different log levels.

Specify the different formats to the constructor with keyword arguments. E.g.:

```
ConditionalFormatter('%(message)s',
                     DEBUG='DEBUG: %(message)s',
                     INFO='just some info... %(message)s')
```

This will use *'%(message)s'* as format for all messages except with level other thand DEBUG or INFO, for which their respective formats are used.

do_format(*record*, *fmt*)

format(*record*)

bibolamazi.core.blogger.**logger = <bibolamazi.core.blogger.BibolamaziLogger object>**
(OBSOLETE) The main logger object. This is a `logging.Logger` object.

Deprecated since version 2.1: This object is still here to keep old code functioning. New code should use the following idiom somewhere at the top of their module:

```
import logging
logger = logging.getLogger(__name__)
```

(Just make sure the logging mechanism has been set up correctly already, see doc for `blogger` module.)

This object has an additional method *longdebug()* (which behaves similarly to *debug()*), for logging long debug output such as dumping the database during intermediate steps, etc. This corresponds to bibolamazi command-line verbosity level 3.

bibolamazi.core.blogger.**setup_simple_console_logging**(*logger=<logging.RootLogger object>*, *stream=<open file '<stderr>', mode 'w'>*)
Sets up the given logger object for simple console output.

The main program module may for example invoke this function on the root logger to provide a basic logging mechanism.

6.6 bibolamazi.core.butils module

Various utilities for use within all of the Bibolamazi Project.

exception bibolamazi.core.butils.**BibolamaziError**(*msg*, *where=None*)
Bases: `exceptions.Exception`

Root bibolamazi error exception.

See also `BibFilterError` and `BibUserCacheError`.

bibolamazi.core.butils.**call_with_args**(*fn*, **args*, ***kwargs*)
Utility to call a function *fn* with **args* and ***kwargs*.

*fn(*args)* must be an acceptable function call; beyond that, additional keyword arguments which the function accepts will be provided from ***kwargs*.

This function is meant to be essentially *fn(*args, **kwargs)*, but without raising an error if there are arguments in *kwargs* which the function doesn't accept (in which case, those arguments are ignored).

bibolamazi.core.butils.**get_version**()
Return the version string `version_str`, unchanged.

bibolamazi.core.butils.**get_version_split**()
Return a 4-tuple *(maj, min, rel, suffix)* resulting from parsing the version obtained via `version.version_str`.

..... TODO: FIXME: CURRENTLY, the elements are strings! why not integers? If not there, they will/should be empty or None?

bibolamazi.core.butils.**getbool**(*x*)
Utility to parse a string representing a boolean value.

If *x* is already of integer or boolean type (actually, anything castable to an integer), then the corresponding boolean convertion is returned. If it is a string-like type, then it is matched against something that looks like 't(rue)?', '1', 'y(es)?' or 'on' (ignoring case), or against something that looks like 'f(alse)?', '0', 'n(o)?' or 'off' (also ignoring case). Leading or trailing whitespace is ignored. If the string cannot be parsed, a `ValueError` is raised.

bibolamazi.core.butils.**guess_encoding_decode**(*dat*, *encoding=None*)

bibolamazi.core.butils.**parse_timedelta**(*in_s*)
Note: only positive timedelta accepted.

bibolamazi.core.butils.**quotearg**(*x*)

bibolamazi.core.butils.**resolve_type**(*typename*, *in_module=None*)
Returns a type object corresponding to the given type name *typename*, given as a string.

..... TODO: MORE DOC .....

bibolamazi.core.butils.**warn_deprecated**(*classname*, *oldname*, *newname*, *modulename=None*, *explanation=None*)

6.7 bibolamazi.core.main module

This module contains the code that implements Bibolamazi's main functionality. It also provides the basic tools for the command-line interface.

class bibolamazi.core.main.**AddFilterPackageAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
Bases: argparse.Action

class bibolamazi.core.main.**ArgsStruct**(*bibolamazifile*, *use_cache*, *cache_timeout*)
Bases: tuple

bibolamazifile
Alias for field number 0

cache_timeout
Alias for field number 2

use_cache
Alias for field number 1

exception bibolamazi.core.main.**BibolamaziNoSourceEntriesError**
Bases: *bibolamazi.core.butils.BibolamaziError*

bibolamazi.core.main.**get_args_parser**()

bibolamazi.core.main.**main**(*argv=['-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)

bibolamazi.core.main.**run_bibolamazi**(*bibolamazifile*, ***kwargs*)

bibolamazi.core.main.**run_bibolamazi_args**(*args*)

bibolamazi.core.main.**setup_filterpackage_from_argstr**(*argstr*)
Add a filter package definition and path to filterfactory.filterpath from a string that is a e.g. a command-line argument to –filterpath or a part of the environment variable BIBOLAMAZI_FILTER_PATH.

bibolamazi.core.main.**setup_filterpackages_from_env**()

bibolamazi.core.main.**verbosity_logger_level**(*verbosity*)
Simple mapping of 'verbosity level' (used, for example for command line options) to correspondig logging level (logging.DEBUG, logging.ERROR, etc.).

6.8 bibolamazi.core.version module

bibolamazi.core.version.**version_str = '3.0'**
The version string. This is increased upon each release.

CHAPTER 7

Python API: Filter Utilities Package

7.1 bibolamazi.filters.util.arxivutil Module

class bibolamazi.filters.util.arxivutil.**ArxivFetchedAPIInfoCacheAccessor**(***kwargs*)
Bases: *bibolamazi.core.bibusercache.BibUserCacheAccessor*

A *BibUserCacheAccessor* for fetching and accessing information retrieved from the arXiv API.

fetchArxivApiInfo(*idlist*)
Populates the given cache with information about the arXiv entries given in *idlist*. This must be, yes you guessed right, a list of arXiv identifiers that we should fetch.

This function performs a query on the arXiv.org API, using the arxiv2bib library. Please note that you should avoid making rapid fire requests in a row (this should normally not happen anyway thanks to our cache mechanism). However, beware that if we get a `403 Forbidden` HTTP answer, we should not continue or else arXiv.org might interpret our requests as a DOS attack. If a `403 Forbidden` HTTP answer is received this function raises *BibArxivApiFetchError* with a meaningful error text.

Only those entries in *idlist* which are not already in the cache are fetched.

idlist can be any iterable.

getArxivApiInfo(*arxivid*)
Returns a dictionary:

```
{
  'reference':  <arxiv2bib.Reference>,
  'bibtex': <bibtex string>
}
```

for the given arXiv id in the cache. If the information is not in the cache, returns *None*.

Don't forget to first call *fetchArxivApiInfo()* to retrieve the information in the first place.

Note the reference part may be a `arxiv2bib.ReferenceErrorInfo`, if there was an error retreiving the reference.

initialize(*cache_obj*, ***kwargs*)

class bibolamazi.filters.util.arxivutil.**ArxivInfoCacheAccessor**(***kwargs*)
Bases: *bibolamazi.core.bibusercache.BibUserCacheAccessor*

A *BibUserCacheAccessor* for fetching and accessing information retrieved from the arXiv API.

complete_cache(*bibdata*, *arxiv_api_accessor*)
Makes sure the cache is complete for all items in *bibdata*.

getArXivInfo(*entrykey*)
Get the arXiv information corresponding to entry citekey *entrykey*. If the entry is not in the cache, returns *None*. Call *complete_cache()* first!

initialize(*cache_obj*, ***kwargs*)

rebuild_cache(*bibdata*, *arxiv_api_accessor*)
Clear and rebuild the entry cache completely.

revalidate(*bibolamazifile*)
Re-validates the cache (with validate()), and calls again complete_cache() to fetch all missing or out-of-date entries.

exception bibolamazi.filters.util.arxivutil.**BibArxivApiFetchError**(*msg*)
Bases: *bibolamazi.core.bibusercache.BibUserCacheError*

bibolamazi.filters.util.arxivutil.**detectEntryArXivInfo**(*entry*)
Extract arXiv information from a *pybtex.database.Entry* bibliographic entry.

Returns upon success a dictionary of the form:

```
{ 'primaryclass': <primary class, if available>,
  'arxivid': <the (minimal) arXiv ID (in format XXXX.XXXX  or  archive/XXXXXXX)>,
  'archiveprefix': value of the 'archiveprefix' field
  'published': True/False <whether this entry was published in a journal other than arxiv>,
  'doi': <DOI of entry if any, otherwise None>
  'year': <Year in preprint arXiv ID number. 4-digit, string type.>
}
```

Note that 'published' is set to True for PhD and Master's thesis. Also, the arxiv.py filter handles this case separately and explicitly, the option there *-dThesesCountAsPublished=0* has no effect here.

If no arXiv information was detected, then this function returns None.

bibolamazi.filters.util.arxivutil.**get_arxiv_cache_access**(*bibolamazifile*)

bibolamazi.filters.util.arxivutil.**setup_and_get_arxiv_accessor**(*bibolamazifile*)

bibolamazi.filters.util.arxivutil.**stripArXivInfoInNote**(*notestr*)
Assumes that notestr is a string in a note={} field of a bibtex entry, and strips any arxiv identifier information found, e.g. of the form 'arxiv:XXXX.YYYY' (or similar).

7.2 bibolamazi.filters.util.auxfile Module

Utilities (actually for now, utility) to parse .aux files from LaTeX documents.

bibolamazi.filters.util.auxfile.**get_all_auxfile_citations**(*jobname*, *bibolamazifile*, *filtername*, *search_dirs=None*, *callback=None*, *return_set=True*)
Get a list of bibtex keys that a specific LaTeX document cites, by inspecting its .aux file.

Look for the file `<jobname>.aux` in the current directory, or in the search directories *search_dirs* if given. Parse that file for commands of the type `\citation{..}`, and collect all the arguments of such commands. These commands are generated by calls to the `\cite{}` command in the LaTeX document.

This effectively gives a list of entries that a particular document cites.

Note: latex/pdflatex must have run at least once on the document already.

CHAPTER 8

Credits, Copyright and Contact information

8.1 Copyright

Bibolamazi is developed and maintained by Philippe Faist. It is distributed under the GNU General Public License (GPL), Version 3 or higher.

8.2 Credits and Third-Party Code

This project also contains the following 3rd party code.

PybTeX is used as python library for parsing and writing BibTeX files.

```
Copyright (c) 2006, 2007, 2008, 2009, 2010, 2011 Andrey Golovizin
```

Full copyright notice is availble in this repo in the file *3rdparty/pybtex/COPYING*.

Arxiv2Bib is a tool for querying the arxiv.org API for preprint details and for parsing its results.

```
Copyright (c) 2012, Nathan Grigg
```

Full copyright notice is available in this repo in the first lines of the file `3rdparty/arxiv2bib/arxiv2bib.py`.

8.3 Contact

Please contact me for any bug reports, or if you want to contribute.

philippe.faist@bluewin.ch

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

b

A

B

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W