

---

# **Bibolamazi Documentation**

*Release 4.6bo*

**Philippe Faist**

**May 28, 2023**



## Contents

<b>1</b>	<b>Introduction to Bibolamazi</b>	<b>3</b>
1.1	What Bibolamazi Does . . . . .	3
1.2	Graphical Application and Command-Line . . . . .	3
1.3	Example Usage Scenario . . . . .	3
1.4	Getting started . . . . .	4
1.5	The Bibolamazi Configuration Section . . . . .	4
1.6	Teaser: Features . . . . .	5
<b>2</b>	<b>Downloading and Installing Bibolamazi</b>	<b>7</b>
2.1	The Bibolamazi Application . . . . .	7
2.2	Installing the Command-Line Interface . . . . .	7
<b>3</b>	<b>Using the Bibolamazi Application</b>	<b>9</b>
3.1	Bibolamazi Operating Mode . . . . .	9
3.2	Getting Started . . . . .	9
3.3	Editing the configuration section . . . . .	10
<b>4</b>	<b>Bibolamazi's configuration section</b>	<b>11</b>
4.1	Syntax . . . . .	11
4.2	Specifying sources . . . . .	11
4.3	Specifying filters . . . . .	12
4.4	Example/Template Configuration Section . . . . .	12
4.5	Available Filters . . . . .	13
4.6	Filter Packages . . . . .	13
<b>5</b>	<b>Using Bibolamazi in Command-Line</b>	<b>15</b>
5.1	First Steps With Bibolamazi Command-Line . . . . .	15
5.2	Bibolamazi Operating Mode . . . . .	15
5.3	The Bibolamazi Configuration Section . . . . .	16
5.4	Content of the Configuration Section . . . . .	16
5.5	Example Full Bibolamazi File . . . . .	16
5.6	Querying Available Filters and Filter Documentation . . . . .	17
5.7	Specifying Filter Packages . . . . .	18
<b>6</b>	<b>Filter Packages</b>	<b>19</b>
6.1	Importing a filter package . . . . .	19
6.2	Creating a new filter package . . . . .	21
6.3	Writing a Simple Filter . . . . .	22
6.4	Writing a New Filter (Full Version) . . . . .	24
<b>7</b>	<b>Python API: Core Bibolamazi Module</b>	<b>31</b>

7.1	Module contents . . . . .	31
7.2	Subpackages . . . . .	31
7.3	bibolamazi.core.argparseactions module . . . . .	48
7.4	bibolamazi.core.bibolamazifile module . . . . .	50
7.5	bibolamazi.core.blogger module . . . . .	56
7.6	bibolamazi.core.butils module . . . . .	58
7.7	bibolamazi.core.main module . . . . .	59
7.8	bibolamazi.core.version module . . . . .	60
<b>8</b>	<b>Python API: Filter Utilities Package</b>	<b>61</b>
8.1	bibolamazi.filters.util.arxivutil Module . . . . .	61
8.2	bibolamazi.filters.util.auxfile Module . . . . .	63
<b>9</b>	<b>Credits, Copyright and Contact information</b>	<b>65</b>
9.1	Copyright . . . . .	65
9.2	Contact . . . . .	65
<b>10</b>	<b>Indices and tables</b>	<b>67</b>
<b>11</b>	<b>Version</b>	<b>69</b>
	<b>Python Module Index</b>	<b>71</b>
	<b>Index</b>	<b>73</b>

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.

Quick links:

- [What is bibolamazi?](#)
- [Download latest release](#)
- [Download & installation instructions](#)
- [Github repository](#)

Table of Contents:



## Introduction to Bibolamazi

### 1.1 What Bibolamazi Does

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 your 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}
```

### 1.2 Graphical Application and Command-Line

Bibolamazi comes in two flavors: a graphical application and a command-line interface. If you’re unsure, go for the graphical application. To install it, download the version suitable for your system (Mac OS X or Windows) on the [github releases page](#).

You can install the command-line version using `pip install bibolamazi`. On Linux, you can also install the graphical application via `pip install bibolamazigui`. For more information, see [Downloading and Installing Bibolamazi](#).

### 1.3 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 unfortunately 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.

## 1.4 Getting started

The bibolamazi application provides an interactive way of creating a new bibolamazi file that will guide you through a standard selection of clean-ups that you can choose from. Open the app, click on `Create New Bibolamazi File ...`, and follow the instructions.

## 1.5 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.

Sources are the normal bibtex files from which bibtex entries are read. A source is specified using the bibolamazi command

```
src: source-file.bib [ alternative-source-file.bib ... ]
```

Alternative source locations can be specified, in case the first file does not exist. This is convenient to locate a file which might be in different locations on different computers. Each source file name can be an absolute path or a relative path (relative to the bibolamazi file). It can also be an HTTP URL which will be downloaded automatically.

You can specify several sources by repeating the `src:` command.



```
src: first-source.bib  alternative-first-source.bib
src: second-source.bib
...
```

Remember: the first readable source of each source command will be read, and merged into the bibolamazi file.

Filters are rules to apply on the whole bibliography database. Their syntax is

```
filter: filter_name <filter-options>
```

The filter is usually meant to deal with a particular task, such as for example changing all first names of authors into initials.

For a list of filters and what they do, please refer the first page of this help browser.

You can usually fine-tune the behavior of the filter by providing options. For a list of options for a particular filter, please refer again to the help page of that filter.

## 1.6 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!



## 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 straightforward 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 3 (this is there by default on most linux and Mac systems).

Additionally, the graphical user interface requires [PyQt5](#). You can do this usually with `pip install pyqt5`. If you're on a linux distribution, it's most probably in your distribution packages. Note you only need PyQt5 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:

```
> cd somewhere/where/I/want/to/keep/bibolamazi/
...> git clone https://github.com/phfaist/bibolamazi
```

- 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 separately. 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
```

## 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 your 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 Getting Started

The easiest way to get started is to open the bibolamazi graphical application, and click on “Create new bibolamazi file”. A sequence of prompts will allow you to specify where your source bibtex files are, and how you’d like to standardize/normalize/transform those entries. Finally, you will be asked where to save the new bibolamazi file. Then, the file will be opened for you.

The code that is displayed is the bibolamazi configuration section. The Bibolamazi program reads this configuration section and parses the corresponding instructions in order to fetch and perform the necessary tasks on your bibliography entries. The new file will already contain a valid configuration section.

Try the “Run Bibolamazi” button at the top: This should run the given filters on the source files you have specified. In order to view the run-time messages, click on the “Messages” tab on the lower part of the window.

You can view the resulting, processed bibtex entries in the “Preview Bibtex Entries” tab. The “Bibolamazi File Info” tab contains some general information about the parsed configuration section.

You may edit the configuration section if you are not happy with the results. In the configuration section, lines that start with `filter:` instruct bibolamazi to apply a filter to all the bibliography entries. The word

immediately after `filter:` (for instance, `filter: arxiv`) specifies which filter to apply. The rest of the line are options which alter the filter's behavior.

For instance, click on a line that starts with `filter:`. The right part of the screen will change so that you can select different options for that filter. If you click on an option in the table, you will see a description of what that option does below the table. Double-click on the right part of the table to edit an option's value.

Don't forget to include the bibolamazi file in place of your bibliography in your LaTeX document. So, if the bibolamazi file is named `main.bibolamazi.bib`, include a line like so in your LaTeX document:

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

## 3.3 Editing the configuration section

For precise instructions on the configuration section, see [Bibolamazi's configuration section](#).

## Bibolamazi's 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.

### 4.1 Syntax

The configuration section should contain instructions of the form:

```
keyword: <instruction>
```

The possible keywords are `src:`, `package:`, and `filter:`, and they should appear in this order in the bibolamazifile (sources must precede package imports which must precede filter instructions).

You may also include comments in the configuration section. Any line starting with two percent signs `%%` will be ignored by bibolamazi:

```
%% This line is a comment and will be ignored by bibolamazi.
```

Comments must be on a line of their own.

### 4.2 Specifying sources

Sources are where your original bibtex entries are stored, i.e., the bibtex entries you would like to process. This can be, for instance, 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 for the same `src:` instruction, only the FIRST file that exists is read. This is useful for example if you often work from two different computers, on which your bibtex source is in different places:

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

You may use separate `src:` instructions to merge entries from several files:

```
src: mysource1.bib
src: anothersource.bib
```

You can also specify alternate locations for each source file:

```
%% merge entries from source1.bib and src2.bib which reside either in %%  
/home/philippe or /Users/philippe src: /home/philippe/bibtexfiles/source1.bib  
/Users/philippe/bibtexfiles/source1.bib src: /home/philippe/bibtexfiles/src2.bib  
/Users/philippe/bibtexfiles/src2.bib
```

Instead of a file name, you may also specify a HTTP or FTP URL. If your filename or URL contains spaces, enclose the name in double quotes: "My Bibtex Library.bib". You can also specify a path like ~/bibtexfiles/mylibrary.bib which is interpreted relative to your HOME directory.

### 4.3 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 -sMode=strip -sUnpublishedMode=eprint
```

In the Bibolamazi application, when editing filter options you can click on the “? info” button to get information about that filter.

If you are using the command-line bibolamazi program, a full list of options can be obtained with:

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

and 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 -sOption1=val1 ...
```

### 4.4 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.
```

(continues on next page)



(continued from previous page)

```
%% **** 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
```

## 4.5 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.

## 4.6 Filter Packages

Filters are organized into filter packages. All built-in filters are in the package named `bibolamazi.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 regular Python package whose modules can be run as filters. These are documented at greater length in the Section [Filter Packages](#). If you develop your own filters, it is recommended to group them in your own filter package (please do not fiddle with the built-in filter package unless you plan to submit your changes to improve bibolamazi).

You can include filter packages from within a bibolamazi file by using the syntax:

```
package: path/to/filter/pkgname
```

The path should point to a directory which is a valid python package, i.e., which contains the `__init__.py` file. More generally, you can use the syntax:

```
package: <filter-package-specification>
```

Where `<filter-package-specification>` is any valid specification of a filter package as documented in [Importing a filter package](#). For instance, since Bibolamazi 4.2, you may also specify a github repository directly:

```
package: github:phfaist/mybibolamazifilters
```

When such a directive is encountered, the package is automatically downloaded in a cached directory, and the filters it contains can directly be used in the bibolamazi file. To specify a specific branch or commit ID, you may use the syntax:

```
package: github:phfaist/mybibolamazifilters/mybranch
```

See [Filter Packages](#) for further documentation on filter packages, how to import them, how to create your own, etc. There, you'll also see how it is possible to register filter packages at specific locations in a way which applies to all bibolamazi files, without having to include package: directives (but then it might be harder to share your bibolamazi file with others).



## Using Bibolamazi in Command-Line

### 5.1 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 myfile.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 various test files provided in the source code. To compile them, run:

```
> bibolamazi 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>
```

### 5.2 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}
```

## 5.3 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
```

## 5.4 Content of the Configuration Section

The content of the configuration section is the same as described in [Bibolamazi's 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).

## 5.5 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.
```

(continues on next page)

(continued from previous page)

```
%
% 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 ...
```

## 5.6 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 `bibolamazi.filters':

arxiv      ArXiv clean-up filter: normalizes the way each bibliographic
           entry refers to arXiv IDs.
citearxiv  Filter that fills BibTeX files with relevant entries to cite
           with \cite{1211.1037}
citeinspirehep Filter that fills BibTeX files with relevant entries to cite
           with e.g. \cite{inspire:PhysRev.47.777--EPR+paper}
citekey    Set the citation key of entries in a standard format
duplicates Produces LaTeX rules to make duplicate entries aliases of one
           another.
echo       Echo a custom message in the bibolamazi log
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.
```

(continues on next page)

(continued from previous page)

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.

---

**Note:** The `--list-filters` option must be given after any `--filterpackage` options.

---

## 5.7 Specifying Filter Packages

The command-line `bibolamazi` by default only knows the built-in filter package `bibolamazi.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'
```

or using the alternative syntax:

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

The argument to `--filter-package` is of the form `'packagename=/path/to/the/filter/package'` or `'/path/to/filter/package'`. Note that in the first syntax, 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; in the second syntax, the path should point to the python package directory itself, i.e., the directory which contains the `__init__.py` file.

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 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 (i.e., a key-value pair `pkgname=/path/for/import`, or `/path/to/filter/pkgname`). In the environment variable, the first given filter packages are searched first.

## Filter Packages

Bibolamazi filters are organized in filter packages. These are regular python packages whose modules can be invoked as filters. This is a directory containing a `__init__.py` file (which defines the directory as a python package) along with python files that define filters. (The `__init__.py` file is usually empty.)

All built-in filters are part of the filter package `bibolamazi.filters`. Additional filter packages can be imported for instance using the `package:` directive (see [package: directive](#)).

It is very simple to create your own filter packages and provide your own filters. This section covers how filter packages can be imported, how to create your own filter packages, and how to write your own filters.

### 6.1 Importing a filter package

Filter packages are imported by providing a filter specification at one of several places in the bibolamazi app or command-line tool.

#### 6.1.1 How to import a package

You have different ways to import a filter package to make the corresponding filters available for use in your bibolamazi file:

- Use a `package:` directive in your bibolamazi file. Along with your sources and filters, specify something like:

```
package: <filter-package-specification>
```

To import the corresponding filter package. See below for possible values of `<filter-package-specification>`.

- (In the bibolamazi application only:) Open the settings dialog (from the main window). Then select the local filter packages tab, and click on the button “Add filter package” to specify your local filter package. You should specify the directory that defines the python package, i.e., the directory that contains the `__init__.py` file.
- You may globally enable and import a filter package by setting the environment variable `BIBOLAMAZI_FILTER_PATH`. This variable may be set to a list of filter packages separated by `:` on Unix/Mac systems and by `;` on Windows systems (as for the `PATH` environment variable).
- (In the bibolamazi command-line tool only:) Use the option `--filter-package=<filter-package-specification>`

After importing a filter package, you can then use all the filters defined in that package in your bibolamazi file. You can also refer to a filter in a specific package, in case two filters from different packages have the same name, with the syntax `mypackage:filtername`. For instance:

```
filter: mypackage:filtername -dOption1 -sOption2=default ...
```

### 6.1.2 Filter Package Specification

A filter specification, indicated by `<filter-package-specification>` above, is a string of one of the following forms:

- `/path/to/some/package` — if a path is given, this must be a folder that defines a Python package (ie., which contains an `__init__.py` file). This package is the filter package.
- `packagename=/some/path/` or `packagename=` — using this syntax, you specify a python package name and the path that needs to be added to the `PYTHONPATH` in order to load that package. For instance, if you have a Python package located at `/home/me/python/packages/pkga` (which contains an `__init__.py` file), you would specify `pkg=/home/me/python/packages`.
- `github:<user or org>/<repo>[/<branch>]` — specify a filter package as a remote github repository. The branch name is optional, and you can use a tag name or commit ID instead of a branch name. Examples:

```
package: github:phfaist/mybibolamazifilters
```

```
package: github:phfaist/mybibolamazifilters/mybranch
```

```
package: github:phfaist/mybibolamazifilters/4c84fd92ec9189ebf28ebd30916d3c9c9e53a8fb
```

When a github repository is specified, the repository must either be the contents of the filter package (it contains a `__init__.py` file directly in the root directory of the repository), or it must contain a folder of the same name as the repo (possibly with hyphens converted to underscores) which is assumed to be the python filter package.

New in version 4.2: Remote github repositories can be specified and automatically accessed since Bibolamazi 4.2

### 6.1.3 Security Considerations for Remote Packages

Downloading remote filter packages presents a security risk, because filters are python scripts that can execute arbitrary code. For this reason, a warning is displayed to the user the first time you access a remote repository. You should acknowledge this risk and remember to be careful.

You're asked once only, and the setting is then stored in the configuration file. You can modify the configuration file directly if you would like to change this setting. The configuration file resides in a system-dependent location which is typically `~/config/bibolamazi/` on Unix-like systems, `C:\Users\<username>\AppData\Local\bibolamazi\` on Windows, and `~/Library/Application Support/bibolamazi/` on Macs; we use the `appdirs` Python package to determine this.

### 6.1.4 Github Authentication for Private Repositories

It's a good idea to set up github authentication if you use github remote packages, as you have two main advantages:

- you can access your private repositories;
- the github API server has higher rate limits and you will be much less likely to reach their query limits.



Bibolamazi uses personal access tokens to authenticate to github. You simply create a dedicated personal access token for bibolamazi by visiting <https://github.com/settings/tokens> and specify the resulting token to bibolamazi. The token can be revoked at any later time and bibolamazi never sees your password.

In the bibolamazi application, go to Settings → Remote filter packages tab, and turn on Use authentication. This will prompt you with specific settings to carry out to generate your personal access token and provide it to bibolamazi.

In the command-line application, run `bibolamazi --github-auth`. This will enter interactive setup mode where instructions are provided on how to generate the access token and provide it to bibolamazi. To un-set any previously set authentication data, use `bibolamazi --github-auth=-`.

New in version 4.2: Remote github repositories can be specified and automatically accessed since Bibolamazi 4.2

### 6.1.5 Local Caches for Remote Packages

The code for remote filter packages (github package specifications) is downloaded and stored in a local cache dir (for instance, this path is `~/Library/Caches/bibolamazi` on Mac OS X). Github repos are checked for updates when the package is requested, unless a check has been performed recently already (currently, this timeout is set to 10 minutes).

If you'd like to force a refresh and re-download a package, you can clear bibolamazi's local package cache by simply removing the entire bibolamazi cache directory (of course, this will also clear the cache for other downloaded packages and they will be refreshed, too).

## 6.2 Creating a new filter package

Creating a filter package is as easy as creating a directory and putting an empty `__init__.py` file in there. That's it. That's your minimal Python package.

For instance, let us create a filter package called `mypackage`. Create a folder called `mypackage`. Then create an empty file called `__init__.py` in that folder. Instead of `mypackage`, you may use the name that you like. Keep in mind, though, that filter package names must be valid Python identifiers, so you need to avoid hyphens, spaces, and accents; you should only use letters, underscores and digits, and the name can't start with a digit.

Then you can put other python files in the package (say `myfirstfilter.py` and `mysecondfilter.py`), which define filters. The Python file name without the `.py` extension is the name of the filter that you can use in the bibolamazi file. For instance, we could have the following file structure:

```
mypackage/
- __init__.py      (empty file)
- myfirstfilter.py (definition of filter myfirstfilter)
- mysecondfilter.py (definition of filter mysecondfilter)
```

You can then import this filter package in your bibolamazi file using, for instance, the directive:

```
package: /path/to/mypackage
```

You can then use `myfirstfilter` and `mysecondfilter` as normal filters in your bibolamazi file. In the following sections, we'll detail how to write custom filters so that they can do useful stuff, i.e., we'll see how we should go about to code the contents of `myfirstfilter.py` and `mysecondfilter.py`.

### 6.2.1 Distributing the filter package as a github repository

You can share your filters by creating a github repository to share your filters. The repository must be structured in one of two ways:

- The repository itself contains a folder which is the python package, which itself contains the `__init__.py` file. The python package must have the same name as the repository, with hyphens converted to underscores.
- The repository may be the python package itself, i.e., there is a `__init__.py` file at the root of the repository.

Create the repository in this way, and then others can use your filters automatically by including the directive:

```
package: github:username/repo
```

You can even keep the repository private, and allow access to your friends; if your friends configure github authentication within bibolamazi [as explained here](#), then they can directly access your filters with the same directive.

## 6.3 Writing a Simple Filter

There are two ways of writing filters. The first one, presented here, is the quick-and-easy way, where you basically only have to define a function. The more elaborate way gives you additional control about more stuff you can do, but requires a little more coding, and will be covered in the section [Writing a New Filter \(Full Version\)](#).

First of all, before create a custom filter, you should [create your own filter package](#). The python files here will refer to individual python files within the filter package that you will have created.

In you custom filter package, create a python file with the name of your future filter and the extension `.py`. In this example, we'll call it `add_constant_field.py`, because our filter will simply add a field with a constant value to all the bibtex entries.

The file `add_constant_field.py` might look like this:

```
# add_constant_field.py:

from pybtex.database import Entry, Person

import logging
logger = logging.getLogger(__name__)

def bib_filter_entry(entry, field_name='note', value='Hello world'):
    """
    Author: Philippe Faist (C) 2019 GPL 3+

    Description: Add a fixed field to each entry

    This filter adds to each entry an additional field named `field_name`
    set to the constant value `value`.

    Arguments:

    * field_name : the field name to insert (or replace) in all
                  bibtex entries
    * value :     the value to set the given field_name to, in
                  all bibtex entries
    """
```

(continues on next page)

(continued from previous page)

```
# write debug messages, which are seen in verbose mode
logger.debug("Taking care of entry %s", entry.key)

# set the field field_name to the given value:
entry.fields[field_name] = value
```

There are two possible ways a filter can act on a bibliography database. Either it can filter all entries individually, where each fix for each entry does not depend on any other entry; or the filter can act on the database as a whole. For instance, a filter that fixes URLs or creates author initials would act entry-by-entry, whereas a duplicates detector would act on the database as a whole.

In the above example, we act on each entry individually, because we defined a function called `filter_bib_entry()`. This function must take a first argument called `entry`. This will be the entry to possibly modify. Any additional arguments to the function are automatically scanned and set according to options of the form `-sKey=Value` and `-dKey` in the `bibolamazi` file. You can see this by clicking on the “? info” button to open you own filter’s help page (in the `bibolamazi` application), or by running `bibolamazi --filterpackage=/path/to/mypackage --help add_constant_field` (in the `bibolamazi` command-line tool).

In the above example, we modify the entry’s fields to add a field with the given field name and field value. For instance, we could use this filter as it is in a `bibolamazi` file with the directives:

```
package: /path/to/mypackage

filter: add_constant_field -sFieldName=annotate
        -sValue='automatic annotation added by my filter'
```

with the effect of adding the field `annotate = {automatic annotation added by my filter}` to all `bibtex` entries.

The function `bib_filter_entry()` may also take an argument called `bibolamazifile` to access properties of the `bibolamazifile`. The argument `bibolamazifile` would then be a `bibolamazi.core.bibolamazifile.BibolamaziFile` instance.

The entry argument is an object of type `pybtex.database.Entry`. See more in the [pybtex documentation](#).

An inconvenience of defining the `bib_filter_entry()` as above in the “simple filter” definition method is that you can’t pre-process the user’s options once before filtering all entries. Because this function will be called many times, it might be necessary in certain cases to perform some task once only, compute some value or get some data, and then filter all entries using this data. For this, you should change to a filter-class based filter module, as described in the next section.

If your filter is supposed to act on the whole bibliography database in one go, then you should define the function `bib_filter_bibolamazifile()` instead of `bib_filter_entry()`. For instance, we could define a filter `remove_books.py` as follows:

```
# remove_books.py:

from pybtex.database import Entry, Person

import logging
logger = logging.getLogger(__name__)

def bib_filter_bibolamazifile(bibolamazifile):
    """
    Author: Philippe Faist, (C) 2019, GPL 3+

    Description: Remove all entries that are of type 'book'

    I have no idea why you'd want to do this, but it provides a nice example
```

(continues on next page)

(continued from previous page)

```
of how to write a filter that acts on the full bib database.
"""

bibdata = bibolamazifile.bibliographyData()

keys_for_removal = []

for key, entry in bibdata.entries.items():
    if entry.type == 'book':
        # mark this key for removal from database
        keys_for_removal.append(key)

# remove entries only after we've done iterating the database
for key in keys_for_removal:
    del bibdata.entries[key]
```

In this example, we iterate over the full bibliography database and remove all entries that are of the type book.

The argument `bibolamazifile` is a `bibolamazi.core.bibolamazifile.BibolamaziFile` instance.

You should proceed by trial and error, and you can get inspired by the existing built-in filters, see <https://github.com/phfaist/bibolamazi/tree/master/bibolamazi/filters>.

Continue reading [Writing a New Filter \(Full Version\)](#) for more in-depth information about how bibolamazi filters work. Really, the “easy” filter definitions presented here are a convenient shorthand for defining a full filter class as described in the next section.

## 6.4 Writing a New Filter (Full Version)

Here we document how filters work in their full glory. For a quick start, you might want to read first [how to develop a quick-n-dirty filter](#).

### 6.4.1 Example of a custom filter

```
#
# bibolamazi filter example
#

from __future__ import print_function, absolute_imports, unicode_literals

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 ---
```

(continues on next page)

(continued from previous page)

```

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

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

HELP_TEXT = r"""
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):
        r"""
        Constructor method for TestFilter.

```

(continues on next page)

(continued from previous page)

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)
```

```
# debug log messages are seen by the user in verbose output mode
logger.debug('my filter constructor: mode=%s, uppercase=%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'] = (
        "On d\u00E9daigne volontiers un but qu'on n'a pas "
        "r\u00E9ussi \u00E0 atteindre, ou qu'on a atteint "
        "d\u00E9finitivement. (Proust)"
```

(continues on next page)

(continued from previous page)

```

    )
    else:
        #
        # Errors can be reported by raising exceptions
        #
        raise BibFilterError('testfilter',
                              "Unknown operating mode: {}".format(mode))

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

    return

#
# Every Python module which defines a filter (except for "easy-style"
# filters) should have the following method. This returns the filter
# class type, which is expected to be a `BibFilter` subclass.
#
def bibolamazi_filter_class():
    return MyTestFilter

```

## 6.4.2 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!

## 6.4.3 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.

### 6.4.4 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.

---

### 6.4.5 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 mention of the contributor with license. E.g.:

`Philippe Faist, (C) 2013, GPL 3+`

- `helpdescription` is a brief description of what the filter does. This is displayed at the top of the filter help page and in the detailed filter lists.
- `helptext` is a long description of what the filter exactly does, how to use it, the advantages, tricks, pitfalls, etc. Make this as detailed as necessary. It is recommended to wrap the text to 80 columns.

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.

### 6.4.6 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.

It is important to specify the type of arguments because this will influence the type of widget that the user will be presented with in the graphical user interface. For instance, if the argument is a bool, then a “True/False” choice dropdown menu is presented, but if the argument is a string, a text input widget is shown.

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

### 6.4.7 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.



## Python API: Core Bibolamazi Module

### 7.1 Module contents

Core bibolamazi module.

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

### 7.2 Subpackages

#### 7.2.1 `bibolamazi.core.bibfilter` package

Module `bibolamazi.core.bibfilter`

**class** `bibolamazi.core.bibfilter.BibFilter(**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. Sets the `filtername` to the name of the filter class.

A warning is emitted for each argument in `kwargs`, which presumably was ignored by the super-class and passed on.

**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 `bibolamazifile` 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` (preferably, 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` (preferably, 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 filter. 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.

**postrun(bibolamazifile)**

This function gets called immediately after the filter is run, before any further filters are 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 global post-processing task should be done immediately after the actual filtering of the data.

You can use this function, e.g., to produce an aggregated warning or report message.

This method is not called if the filter raised an exception, whether internal or not.

The default implementation does nothing.

**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 pre-processing 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.

**bibolamazi.core.bibfilter.argtypes module****class bibolamazi.core.bibfilter.argtypes.ColonCommaStrDict(\*args, \*\*kwargs)**

Bases: dict

A dictionary of values, specified as a comma-separated string of pairs 'key:value'. If no value is given (no colon), then the value is None.

**type\_arg\_input** = <bibolamazi.core.bibfilter.argtypes.StrEditableArgType object>

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

Bases: list

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

**type\_arg\_input** = <bibolamazi.core.bibfilter.argtypes.StrEditableArgType object>

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

Bases: object

**option\_val\_repr(x)**

**class bibolamazi.core.bibfilter.argtypes.LogLevel(val=None)**

Bases: object

One of 'CRITICAL', 'ERROR', 'WARNING', 'INFO', 'DEBUG', or 'LONGDEBUG'.

**levelnos** = [('CRITICAL', 50), ('ERROR', 40), ('WARNING', 30), ('INFO', 20), ('DEBUG', 10), ('LONGDEB

**levelnos\_dict** = {'CRITICAL': 50, 'DEBUG': 10, 'ERROR': 40, 'INFO': 20, 'LONGDEBUG': 5, 'WARNING': 30

**levelnos\_list** = ['CRITICAL', 'ERROR', 'WARNING', 'INFO', 'DEBUG', 'LONGDEBUG']

**type\_arg\_input** = <bibolamazi.core.bibfilter.argtypes.EnumArgType object>

**class bibolamazi.core.bibfilter.argtypes.MultiTypeArgType(typelist, parse\_value\_fn)**

Bases: object

**option\_val\_repr(x)**

**class bibolamazi.core.bibfilter.argtypes.StrEditableArgType**

Bases: object

**option\_val\_repr(x)**

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

Define a class with the given name `class_name`, which can store one of several choices. The possible values are fixed. Each choice has an integer and a string representation.

`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.argtypes.multi_type_class(class_name, typelist,
                                                    value_attr_name='value', val-
                                                    uetype_attr_name='valuetype',
                                                    convert_functions=[(<class
                                                                    'bool'>,
                                                                    <function
                                                                    getbool>)],
                                                    parse_value_fn=None, doc=None)
```

Define a class with the given name `class_name`, which can store a value of one of several fixed types. This is used, for instance, in the `fixes` filter where for some options one can specify either “True/False” (bool type) or a list of fields (CommaStrList type).

`class_name` is the class name.

`typelist` should be a list of tuples (`typeobject`, `description`) of type objects that can be stored by this object and a corresponding very short description of what is stored with that type

`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.

`valuetype_attr_name` the name of the attribute in the class that should store the type object that is currently stored.

Optionally, you can also specify a list of helper functions that can convert stuff into a given type: `convert_functions` is a list of tuples (`type_object`, `function`) that specifies this.

If `parse_value_fn` is not `None`, then it should be set to a callable that parses a value and returns a tuple (`typeobject`, `value`). It can raise `ValueError`.

**bibolamazi.core.bibfilter.factory module**

```
class bibolamazi.core.bibfilter.factory.DefaultFilterOptions(filtername=None, filter-
                                                            path={'bibolamazi.filters':
                                                            None}, finfo=None)
```

Bases: `object`

Instantiate this class as:

```
defopt = DefaultFilterOptions(filtername [, filterpath=...]) OR
defopt = DefaultFilterOptions(finfo=...)
```

**filterAcceptsVarArgs()**

Returns True if the filter can accept additional positional arguments.

**filterAcceptsVarKwargs()**

Returns True if the filter can accept additional keyword arguments.

**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 were 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).

NOTE: This function doesn't actually validate all options to check whether the filter will accept them (e.g., options like `'-sKey=Value'` will be blindly appended to the `kwargs`). See `FilterInfo.validateOptionStringArgs()` for that.

**parser()**

**use\_auto\_case()**

**class** `bibolamazi.core.bibfilter.factory.FilterArgumentParser`(`filtername`, `**kwargs`)

Bases: `argparse.ArgumentParser`

**error**(`message: string`)

Prints a usage message incorporating the message to `stderr` and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

**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 to the filter constructor itself raising an exception.

**fmt**(name)

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

Bases: `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)

**class** bibolamazi.core.bibfilter.factory.**FilterInfo**(name, filterpath={'bibolamazi.filters':  
None})

Bases: `object`

Information about a given filter.

Arguments:

- **name**: the name of the filter to get information about. It may be of the form 'filtername' or 'pkgname:filtername'.
- **filterpath**: an `collections.OrderedDict` of paths where to look for filters. Keys are package names and values are the filesystem directory that needs to be added to `sys.path` to be able to import the given filter package as a Python package.

**filtername**

The filter name (without any filter package information).

**fmodule**

The module object that contains the filter.

**fclass**

The class object that implements the filter.

Note that simple-syntax filters are internally translated into a class object that fulfils the full-syntax filters API. So all filters have a valid class object that behaves as one would expect.

**filterpackagename**

The name of the filter package in which this filter is located. This value is always valid. The built-in filters are in the filter package `bibolamazi.filters`.

**filterpackagedir**

The filesystem path that needs to be added to `sys.path` in order to be able to import the filter package `filterpackagename` as a Python package. Note: `bibolamazi` assumes that `sys.path` is not being mischievously manipulated by other parts of the Python code. If the filter package can be loaded from Python's default path, this directory may not be accurate.

**filterpackagespec**

A string of the form 'pkgname=file/system/path' that combines the information of filterpackage-name and filterpackagedir in a single string. This is a valid argument to the function `parse_filterpackage_argstr()`.

This object also exposes the following attributes, which reflect what arguments were given to the constructor.

**name**

The filter name, as specified to the constructor.

**filterpath**

The filter path specified to the constructor (or the default one), that was used to resolve the information to the present filter.

The following methods are available for more specific filter information.

**defaultFilterOptions()**

Return a `DefaultFilterOptions` object that is capable of standard filter argument parsing for this filter.

This method returns `None` if the filter doesn't use the default argument parsing mechanism.

**formatFilterHelp()**

Get the filter's help text.

This is either the filter's own custom help text, or the help text retrieved from the filter's argument parser.

**static initFromModuleObject(filtername, module\_obj, fpkgname, fpkgdir)**

Initializes a `FilterInfo` object from an already-imported module object `module_obj` in a given filter package with name `fpkgname` residing in `dir fpkgdir`. The module name without the package information must be given in `filtername`.

**makeFilter(optionstring)**

Instantiate the filter with the given option string.

Returns the new filter instance.

Raises `py:exc:FilterCreateError` if any exception was raised during the filter instantiation.

**parseOptionStringArgs(optionstring)****validateOptionStringArgs(pargs, kwargs)**

Validate the arguments as OK to pass to constructor, i.e. that all argument names are correct.

We use `inspect.getcallargs()` to inspect the filter class constructor's signature.

Raises `FilterCreateArgumentError` if the validation fails.

**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)**

**class** `bibolamazi.core.bibfilter.factory.FilterPackageSpec`(`a`, `b=None`)

Bases: `object`

Stores a filter package specification, and provides a convenient API to download remote packages.

Constructor: `FilterPackageSpec(argstr)` or `FilterPackageSpec(path)` or `FilterPackageSpec(url)` or `FilterPackageSpec(fpname, fpdir)`

Properties:

- `is_url` – True if the filter package was specified directly as a path to the python package, or as a URL. False if individual filter names and python-path were specified.
- `url` – if `is_url==True`, then this is the path or URL that was specified.
- `fpname` – the name of the filter, may be None if `is_url==True` until `materialize()` is called.
- `fpdir` – the directory which needs to be added to `sys.path` to load the filter package, may be None if `is_url==True` until `materialize()` is called.

New in version 4.2: Added `FilterPackageSpec` class.

**materialize()**

Ensures that the filter package is present on the local filesystem, if it is a remote filesystem, and returns specific information on how to load it.

This method may be called regardless of whether the spec was originally an URL or not, i.e., for either value of `is_url` the return value of this is correct.

Returns a tuple (`fpname`, `fpdir`) of a filter package name and filesystem directory which needs to be added to the `sys.path` in order to load the former.

**repr()**

**exception** `bibolamazi.core.bibfilter.factory.ModuleNotAValidFilter(fname, errorstr=None)`

Bases: `bibolamazi.core.bibfilter.factory.NoSuchFilter`

Signifies that a given module does not expose a valid bibolamazi filter. See also [FilterInfo](#).

**exception** `bibolamazi.core.bibfilter.factory.NoSuchFilter(fname, errorstr=None)`

Bases: `Exception`

Signifies that the requested filter was not found. See also [FilterInfo](#).

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

Bases: `Exception`

Signifies that the requested filter package was not found. See also [FilterInfo](#).

**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(force_redetect=False, filter-path={'bibolamazi.filters': None})`

`bibolamazi.core.bibfilter.factory.detect_filters(force_redetect=False, filter-path={'bibolamazi.filters': None})`

`bibolamazi.core.bibfilter.factory.format_filter_help`(filename=None, filter-  
path={'bibolamazi.filters': None})  
Format help text for the given filter. This is a shortcut to the corresponding method in `FilterInfo`.

`bibolamazi.core.bibfilter.factory.inspect_getargspec`(f)

`bibolamazi.core.bibfilter.factory.load_precompiled_filters`(filterpackage, precom-  
piled\_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, options, filter-  
path={'bibolamazi.filters': None})

Main filter instance factory function: Create the filter instance.

This is a simple wrapper for `FilterInfo(...).makeFilter(...)`.

`bibolamazi.core.bibfilter.factory.package_provider_manager` = None

The package provider manager. If we a filterpackage is specified with a URL, then this is the manager we use to retrieve the remote filter package.

The main module is responsible for instantiating a `PackageProviderManager` instance and storing it here.

`bibolamazi.core.bibfilter.factory.parse_argdoc`(doc)

Parses argument documentation from a docstring. Extracts lists of argument documentation in a relatively crude way. Expects arguments to be documented in lines of the form:

```
* argument_name (type) : Here comes the documentation of the argument. It
    may span several lines, that is ok.
*arg2: as you can see, whitespace is optional and ignored; the type
    name is also not necessary.
- arg3: also, argument listings may begin with a dash instead of an asterisk
```

The argument list is expected to be at the end of the docstring. I.e., anything that follows the argument list will be included in the doc of the last argument!

If there is a line with the single word 'Arguments' (with possible punctuation), e.g.:

```
Arguments:
```

Then argdocs are processed only after that line.

Returns a tuple (argdoclist, fndoc). The argdoclist is a list of objects (actually, named tuples), with the fields 'argname', 'argtypename', and 'doc'. If an argtypename is absent, it is set to an empty value. The fndoc is a string corresponding to the rest of the docstring before the argument documentation.

`bibolamazi.core.bibfilter.factory.parse_filterpackage_argstr`(argstr)

Parse filter package specification given as "path/to/the/package" or as a URL. The format "filterpackage=path/to/the/package" or "filterpackage=" is also accepted for local filesystem paths, but not for URLs.

Returns a tuple (fpname, fpdir) of a filter package name and filesystem directory which needs to be added to the sys.path in order to load the former.

Note that if a remote URL is provided, then it is downloaded and stored in cache.

Deprecated since version 4.2: Use the `FilterPackageSpec` class instead.

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

```
bibolamazi.core.bibfilter.factory.validate_filter_package(fpname, fpdir,
                                                         raise_exception=True)
    Make sure given filter package at given directory is valid.
    Utility to warn the user of invalid -filterpackage option
```

## 7.2.2 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)`

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 understand.

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

**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 comparison 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 comparison 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 understand.

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)

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 comparison 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 understand.

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

```
class bibolamazi.core.bibusercache.tokencheckers.TokenCheckerDate(time_valid=datetime.timedelta(days=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)`

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 comparison 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(**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 understand.



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

**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)

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 comparison 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.

**has\_entry\_for**(key)

Returns True if we have a token checker set for the given entry key.

**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 understand.

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

**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. Comparison is done with the equality operator `==` (actually using the original `TokenChecker` implementation).

```
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 understand.

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

## 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 considered 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 `cacheAccessor()` (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 `refreshCache()`, `fetchMissingItems(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 `NotImplementedError` 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: `collections.abc.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)**

**items()** → a set-like object providing a view on D's items

#### **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 tokenchecker should be a TokenChecker instance. 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.

**token\_for**(key)

Return the token that was stored associated with the given key.

Raise an exception if no cache validation set or if the key doesn't exist.

**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: `collections.abc.MutableSequence`

**append**(value)

S.append(value) – append value to the end of the sequence

**insert**(index, value)

S.insert(index, value) – insert value before index

## 7.3 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.)

**class** `bibolamazi.core.argparseactions.GithubAuthSetup`(parser)

Bases: `object`

**interactive\_enter**()

**interactive\_manage\_token**()

**interactive\_setup\_token**()

**print\_status**()

**save\_token\_and\_exit**(token)

```
    setup_from_arg(arg)
    unset_token_and_exit()
class bibolamazi.core.argparseactions.opt_action_github_auth(option_strings, dest,
                                                                nargs=None, const=None,
                                                                default=None, type=None,
                                                                choices=None, re-
                                                                quired=False, help=None,
                                                                metavar=None)
    Bases: argparse.Action
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_helpwelcome(option_strings, dest,
                                                                nargs=None, const=None,
                                                                default=None, type=None,
                                                                choices=None, re-
                                                                quired=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
class bibolamazi.core.argparseactions.store_key_bool(option_strings, dest, nargs=1,
                                                       const=True, exception=<class 'Val-
                                                       ueError'>, **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, excep-
                                                       tion=<class '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
```

## 7.4 bibolamazi.core.bibolamazifile module

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

`bibolamazi.core.bibolamazifile.AFTER_CONFIG_TEXT = '\n%\n% This file was generated by BIBOLAMAZI __BIBOLA`  
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 instantiated and data from sources loaded.  
See doc for `BibolamaziFile`.

`bibolamazi.core.bibolamazifile.BIBOLAMAZIFILE_PARSED = 2`  
Bibolamazi file load state: data read and parsed, filters instantiated 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.

**exception** `bibolamazi.core.bibolamazifile.BibFilterInternalError(fname, filtername, filter_exc, tbmsg)`  
Bases: `bibolamazi.core.butils.BibolamaziError`

**exception** `bibolamazi.core.bibolamazifile.BibolamaziBibtexSourceError(msg, fname=None)`  
Bases: `bibolamazi.core.butils.BibolamaziError`

**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 instantiated. Methods such as `filters()` or `sourceLists()` may be called.
- `BIBOLAMAZIFILE_LOADED`: The bibolamazi file has been read and parsed, filter objects have been instantiated 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.

The constructor creates 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 instantiated 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.

If the cache accessor was not loaded, then `None` is returned.

#### **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()`.

#### **static cacheFileNameFor(fname)**

Return the cache file name corresponding to the given bibolamazi file name.

#### **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`.

**filterPath()**

Return the list of additional filter paths set by the bibolamazi file using ‘package:’ commands.

The returned object is a `PrependOrderedDict`.

See also `fullFilterPath()`.

**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 instantiated 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`.

**fullFilterPath()**

Return the full search path used when creating filter instances from this bibolamazi file.

The returned object is a `PrependOrderedDict`.

See also `filterPath()`.

**getLoadState()**

Returns the state of the `BibolamaziFile` object. One of `BIBOLAMAZIFILE_INIT`, `BIBOLAMAZIFILE_READ`, `BIBOLAMAZIFILE_PARSED`, or `BIBOLAMAZIFILE_LOADED`.

**instantiateFilter(filename, filoptionstring, filterpath=None)**

Look up filter named `filename` (using `filterpath` if specified, or else use `fullFilterPath()`) and instantiate it with the option string `filoptionstring`. Return the filter instance.

The returned filter is not registered in the bibolamazifile’s list of filters, which corresponds to those filters specified in the config section.

**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.

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`.

#### **registerFilterInstance(filterinstance)**

Set up caches, etc., so that the filter can run on this bibolamazifile instance.

#### **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.

#### **runFilter(filter\_instance)**

#### **saveCache(cachefname=None)**

Save the cache. If `cachefname` is `None`, the cache file name is deduced from the current file name (see `fname()`).

The argument `cachefname` is parsed exactly as in the method `saveToFile()`.

---

**Note:** If you call `saveToFile()`, then the cache is automatically saved and a separate call to `saveCache()` is not necessary.

---

Warning: This method will silently overwrite any existing file of the same name.

**saveRawToFile**(fname=None, cachefname=None)

Save the current bibolamazi file object to disk, using the `rawRest()` content instead of the current bib database.

This should only be useful in an editor (e.g., the GUI) where you edit the config and save back to disk without running bibolamazi.

New in version 4.2: Added the `saveRawToFile()` method.

**saveToFile**(fname=None, cachefname=None)

Save the current bibolamazi file object to disk.

This method will write the bibliography data to the file specified to by `fname` (or `fname()` if `fname=None`). Specifically, we will write 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.

The cache is also saved, unless `cachefname=False`. If `cachefname=None` (the default) or `cachefname=True`, the cache is saved to the old file name with extension `‘.bibolamazicache’`, that is, the one given by `self.fname()` and not in `fname`. (The rationale is that we want to be able to use the cache next time we open the file `self.fname()`.) You may also specify `cachefname=<file name>` to save the cache to a specific file name. WARNING: this file is silently overwritten.

**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 cleansifies 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 considered 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 `BIBOLAMAZI-FILE_LOADED`.

Note that you may also use the option in the constructor `default_cache_invalidation_time`, which has the same effect as this function 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-instantiate 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={})
```

Bases: object

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.

## 7.5 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)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using `LogRecord.getMessage()`. If the formatting string uses the time (as determined by a call to `usesTime()`, `formatTime()` is called to format the event time. If there is exception information, it is formatted using `formatException()` and appended to the message.

**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 than DEBUG or INFO, for which their respective formats are used.

```
do_format(record, fmt)
```

```
format(record)
```

Format the specified record as text.

The record’s attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

```
bibolamazi.core.blogger.logger = <BibolamaziLogger bibolamazi.old_logger (WARNING)>
```

(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=<RootLogger root (WARN-
ING)>, stream=<_io.TextIOWrapper
name='<stderr>' mode='w'
encoding='UTF-8'>, level=None,
capture_warnings=True)
```

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.

## 7.6 bibolamazi.core.butils module

Various utilities for use within all of the Bibolamazi Project.

**exception** `bibolamazi.core.butils.BibolamaziError(msg, where=None)`  
Bases: `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_copyrightyear()`  
Return the copyright year `copyright_year`, unchanged.

`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 conversion 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.latex_to_text(x)`

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

`bibolamazi.core.butils.quotearg(x)`  
If `x` contains only non-special characters, it is returned as is. The non-special characters are: all alphanumerical chars, hyphen, dot, slash, colon, tilde, percent, hash. Otherwise, put the value `x` in double-quotes, escaping all double-quotes and backslashes in the value of `x` by a backslash.

The argument `x` may be either a python string or unicode object.

```
For example:      >>> print(quotearg('kosher_name_clean'))    kosher_name_clean
>>> print(quotearg('dirty name with spaces'))    "dirty name with spaces"
>>> print(quotearg(r"reallydirty"
name::with/tons&#$of special chars!!!"))    "really\dirty"
name::with/tons&#$of special chars!!!"
```

```
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, module-
                                     name=None, explanation=None)
```

## 7.7 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, output)
```

Bases: `tuple`

Create new instance of `ArgsStruct(bibolamazifile, use_cache, cache_timeout, output)`

**bibolamazifile**

Alias for field number 0

**cache\_timeout**

Alias for field number 2

**output**

Alias for field number 3

**use\_cache**

Alias for field number 1

```
exception bibolamazi.core.main.BibolamaziNoSourceEntriesError
```

Bases: `bibolamazi.core.butils.BibolamaziError`

```
class bibolamazi.core.main.CmdlMainPackageProviderManager
```

Bases: `bibolamazi.core.bibfilter.pkgprovider.PackageProviderManager`

**remoteAllowed()**

```
class bibolamazi.core.main.CmdlSettings(configfname='cmdl_settings.json')
```

Bases: `object`

Stores settings for the command-line app. Read/write json-objects to the `config` property of this object. Config is loaded upon object creation. Call `saveConfig()` after changing the config property.

**reloadConfig()**

**saveConfig()**

```
bibolamazi.core.main.get_args_parser()
```

```
bibolamazi.core.main.get_github_auth_status()
```

Returns one of `None` (no configuration provided), `False` (configuration exists, token explicitly not set), and `True` (token previously set and saved).

```
bibolamazi.core.main.load_filterpackage_providers()
```

```
bibolamazi.core.main.main(argv=['-T', '-E', '-b', 'latex', '-d', '_build/doctrees', '-D', 'language=en', ' ',
                               '/home/docs/checkouts/readthedocs.org/user_builds/bibolamazi/checkouts/latest/_readth
```

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

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

`bibolamazi.core.main.save_github_auth_token(github_auth_token)`

`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 `-filterpackage` 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 corresponding logging level (`logging.DEBUG`, `logging.ERROR`, etc.).

## 7.8 bibolamazi.core.version module

`bibolamazi.core.version.copyright_year = '2023'`

Year of copyright.

`bibolamazi.core.version.version_str = '4.6b0'`

The version string. This is increased upon each release.



## Python API: Filter Utilities Package

### 8.1 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.

`arxiv_403_received = False`

**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>,
  'error': <None or an error 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 retrieving the reference. In that case, the key 'error' contains an error string.

**initialize**(cache\_obj, \*\*kwargs)

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 `NotImplementedError` exception.

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

Cache accessor for detected arXiv information about bibliography entries.

```
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)
    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 `NotImplementedError` exception.

```
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.>
  'isoldarxivid': <Whether the arXiv ID is of old style, i.e. 'primary-class/XXXXXXX'>
```

(continues on next page)

(continued from previous page)

```
'isnewarxivid': <Whether the arXiv ID is of new style, i.e. 'XXXX.XXXX+' (with 4 or more_
↪digits after dot)>,
}
```

Note that ‘published’ is set to True for PhD and Master’s thesis. Also, the arxiv 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).

## 8.2 bibolamazi.filters.util.auxfile Module

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

```
bibolamazi.filters.util.auxfile.get_action_jobname(jobname, bibolamazifile)
```

If jobname is non-None and nonempty, then return jobname. Otherwise, find the basename of the bibolamazifile, and return that.

New in version 4.3: Added function `get_action_jobname()`.

```
bibolamazi.filters.util.auxfile.get_all_auxfile_citations(jobname, bibolamazifile, filter-
name, search_dirs=None, call-
back=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.

Return a python set (unless return\_set=False) with the list of all bibtex keys that the latex document cites.

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

Arguments:

- **jobname**: the base name of the TEX file; the AUX file that is searched for is “<jobname>.aux”. The jobname is expected to be non-empty; see `get_action_jobname()` for help on that.
- **bibolamazifile**: The bibolamazifile relative to which we are analyzing citations. This is used to determine in which directory(ies) to search for the AUX file.
- **filtername**: The name of the filter that is calling this function. Used for error messages and logs.
- **search\_dirs**: list of directories in which to search for the AUX file. These may be absolute paths or relative paths; the latter are interpreted as being relative to the bibolamazifile’s location.
- **callback**: A python callable can be specified in this argument. It will be called for each occurrence of a citation in the document, with the citation key as single argument.
- **return\_set**: If True (the default), then this function returns a python set with all the citation keys encountered. Set this to False if you’re going to ignore the return value of this function.



## Credits, Copyright and Contact information

### 9.1 Copyright

Copyright (c) 2014 Philippe Faist

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

### 9.2 Contact

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

[philippe.faist@bluewin.ch](mailto:philippe.faist@bluewin.ch)



# 10

## Indices and tables

- `genindex`
- `modindex`
- `search`





*11*

**Version**

Documentation built from bibolamazi version 4.6b0.



## Python Module Index

### b

- `bibolamazi.core`, [31](#)
- `bibolamazi.core.argparseactions`, [48](#)
- `bibolamazi.core.bibfilter`, [31](#)
- `bibolamazi.core.bibfilter.argtypes`, [34](#)
- `bibolamazi.core.bibfilter.factory`, [35](#)
- `bibolamazi.core.biblamazifile`, [50](#)
- `bibolamazi.core.bibusercache`, [45](#)
- `bibolamazi.core.bibusercache.tokencheckers`,  
[41](#)
- `bibolamazi.core.blogger`, [56](#)
- `bibolamazi.core.butils`, [58](#)
- `bibolamazi.core.main`, [59](#)
- `bibolamazi.core.version`, [60](#)
- `bibolamazi.filters.util.arxivutil`, [61](#)
- `bibolamazi.filters.util.auxfile`, [63](#)



## A

action() (bibolamazi.core.bibfilter.BibFilter method), 31  
 add\_entry\_check() (bibolamazi.core.bibusercache.tokencheckers.TokenCheckerPerEntry method), 44  
 AddFilterPackageAction (class in bibolamazi.core.main), 59  
 AFTER\_CONFIG\_TEXT (in module bibolamazi.core.bibolamazifile), 50  
 append() (bibolamazi.core.bibusercache.BibUserCacheList method), 48  
 ArgsStruct (class in bibolamazi.core.main), 59  
 arxiv\_403\_received (bibolamazi.filters.util.arxivutil.ArxivFetchedAPIInfoCacheAccessor attribute), 61  
 ArxivFetchedAPIInfoCacheAccessor (class in bibolamazi.filters.util.arxivutil), 61  
 ArxivInfoCacheAccessor (class in bibolamazi.filters.util.arxivutil), 62

## B

BIB\_FILTER\_BIBOLAMAZIFILE (bibolamazi.core.bibfilter.BibFilter attribute), 31  
 BIB\_FILTER\_SINGLE\_ENTRY (bibolamazi.core.bibfilter.BibFilter attribute), 31  
 BibArxivApiFetchError, 62  
 BibFilter (class in bibolamazi.core.bibfilter), 31  
 BibFilterError, 34  
 BibFilterInternalError, 50  
 bibliographyData() (bibolamazi.core.bibolamazifile.BibolamaziFile method), 51  
 bibliographydata() (bibolamazi.core.bibolamazifile.BibolamaziFile method), 51  
 bibolamazi.core (module), 31  
 bibolamazi.core.argparseactions (module), 48  
 bibolamazi.core.bibfilter (module), 31  
 bibolamazi.core.bibfilter.argtypes (module), 34  
 bibolamazi.core.bibfilter.factory (module), 35

bibolamazi.core.bibolamazifile (module), 50  
 bibolamazi.core.bibusercache (module), 45  
 bibolamazi.core.bibusercache.tokencheckers (module), 41  
 bibolamazi.core.blogger (module), 56  
 bibolamazi.core.butils (module), 58  
 bibolamazi.core.main (module), 59  
 bibolamazi.core.version (module), 60  
 bibolamazi.filters.util.arxivutil (module), 61  
 bibolamazi.filters.util.auxfile (module), 63  
 BIBOLAMAZI\_FILE\_ENCODING (in module bibolamazi.core.bibolamazifile), 50  
 BibolamaziBibtexSourceError, 50  
 BibolamaziConsoleFormatter (class in bibolamazi.core.blogger), 56  
 BibolamaziError, 58  
 bibolamazifile (bibolamazi.core.main.ArgsStruct attribute), 59  
 BibolamaziFile (class in bibolamazi.core.bibolamazifile), 50  
 bibolamazifile() (bibolamazi.core.bibfilter.BibFilter method), 32  
 bibolamazifile() (bibolamazi.core.bibusercache.BibUserCacheAccessor method), 46  
 BIBOLAMAZIFILE\_INIT (in module bibolamazi.core.bibolamazifile), 50  
 BIBOLAMAZIFILE\_LOADED (in module bibolamazi.core.bibolamazifile), 50  
 BIBOLAMAZIFILE\_PARSED (in module bibolamazi.core.bibolamazifile), 50  
 BIBOLAMAZIFILE\_READ (in module bibolamazi.core.bibolamazifile), 50  
 BibolamaziFileCmd (class in bibolamazi.core.bibolamazifile), 55  
 BibolamaziFileParseError, 56  
 BibolamaziLogger (class in bibolamazi.core.blogger), 56  
 BibolamaziNoSourceEntriesError, 59  
 BibUserCache (class in bibolamazi.core.bibusercache), 45  
 BibUserCacheAccessor (class in bibolamazi.core.bibusercache), 46  
 BibUserCacheDic (class in bibolamazi.core.bibusercache), 47

BibUserCacheError, 48  
 BibUserCacheList (class in  
     biblamazi.core.bibusercache), 48

## C

cache\_timeout (biblamazi.core.main.ArgsStruct  
     attribute), 59  
 cacheAccessor() (biblamazi.core.bibfilter.BibFilter  
     method), 32  
 cacheAccessor() (biblamazi.core.biblamazifile.  
     BiblamaziFile method),  
     51  
 cacheDic() (biblamazi.core.bibusercache.  
     BibUserCacheAccessor method),  
     46  
 cacheExpirationTokenChecker() (biblamazi.core.  
     bibusercache.BibUserCache method),  
     45  
 cacheFileName() (biblamazi.core.biblamazifile.  
     BiblamaziFile method),  
     51  
 cacheFileNameFor() (biblamazi.core.biblamazifile.  
     BiblamaziFile static method),  
     51  
 cacheFor() (biblamazi.core.bibusercache.  
     BibUserCache method),  
     45  
 cacheName() (biblamazi.core.bibusercache.  
     BibUserCacheAccessor method),  
     47  
 cacheObject() (biblamazi.core.bibusercache.  
     BibUserCacheAccessor method),  
     47  
 call\_with\_args() (in module biblamazi.core.butils),  
     58  
 checker\_for() (biblamazi.core.bibusercache.  
     tokencheckers.TokenCheckerPerEntry  
     method), 44  
 child\_notify\_changed() (biblamazi.core.  
     bibusercache.BibUserCacheDic method),  
     47  
 CmdlMainPackageProviderManager (class in  
     biblamazi.core.main), 59  
 CmdlSettings (class in biblamazi.core.main), 59  
 cmp\_tokens() (biblamazi.core.bibusercache.  
     tokencheckers.TokenChecker method),  
     42  
 cmp\_tokens() (biblamazi.core.bibusercache.  
     tokencheckers.TokenCheckerCombine  
     method), 42  
 cmp\_tokens() (biblamazi.core.bibusercache.  
     tokencheckers.TokenCheckerDate method),  
     43  
 cmp\_tokens() (biblamazi.core.bibusercache.  
     tokencheckers.TokenCheckerPerEntry  
     method), 44  
 ColonCommaStrDict (class in  
     biblamazi.core.bibfilter.argtypes), 34  
 CommaStrList (class in  
     biblamazi.core.bibfilter.argtypes), 34  
 complete\_cache() (biblamazi.filters.util.arxivutil.  
     ArxivInfoCacheAccessor method),  
     62

ConditionalFormatter (class in  
     biblamazi.core.blogger), 57  
 CONFIG\_BEGIN\_TAG (in module  
     biblamazi.core.biblamazifile), 56  
 CONFIG\_END\_TAG (in module  
     biblamazi.core.biblamazifile), 56  
 configCmds() (biblamazi.core.biblamazifile.  
     BiblamaziFile method),  
     51  
 configData() (biblamazi.core.biblamazifile.  
     BiblamaziFile method),  
     51  
 configLineNo() (biblamazi.core.biblamazifile.  
     BiblamaziFile method),  
     51  
 copyright\_year (in module biblamazi.core.version),  
     60

## D

DefaultFilterOptions (class in  
     biblamazi.core.bibfilter.factory), 35  
 defaultFilterOptions()  
     (biblamazi.core.bibfilter.factory.FilterInfo  
     method), 38  
 detect\_filter\_package\_listings() (in module  
     biblamazi.core.bibfilter.factory), 39  
 detect\_filters() (in module  
     biblamazi.core.bibfilter.factory), 39  
 detectEntryArXivInfo() (in module  
     biblamazi.filters.util.arxivutil), 62  
 do\_format() (biblamazi.core.blogger.  
     ConditionalFormatter method),  
     57

## E

EntryFieldsTokenChecker (class in biblamazi.core.  
     bibusercache.tokencheckers),  
     41  
 enum\_class() (in module  
     biblamazi.core.bibfilter.argtypes), 35  
 EnumArgType (class in  
     biblamazi.core.bibfilter.argtypes), 34  
 error() (biblamazi.core.bibfilter.factory.  
     FilterArgumentParser method),  
     36  
 exit() (biblamazi.core.bibfilter.factory.  
     FilterArgumentParser method),  
     36

## F

fclass (biblamazi.core.bibfilter.factory.FilterInfo  
     attribute), 37  
 fdir() (biblamazi.core.biblamazifile.BiblamaziFile  
     method), 51  
 fetchArxivApiInfo() (biblamazi.filters.util.arxivutil.  
     ArxivFetchedAPIInfoCacheAccessor  
     method), 61  
 fileLineNo() (biblamazi.core.biblamazifile.  
     BiblamaziFile method),  
     52  
 filter\_bibentry() (biblamazi.core.bibfilter.BibFilter  
     method), 32

- `filter_biblamazifile()`  
(`biblamazi.core.bibfilter.BibFilter` method),  
32
  - `filterAcceptsVarArgs()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
35
  - `filterAcceptsVarKwargs()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
35
  - `FilterArgumentParser` (class in  
    **biblamazi.core.bibfilter.factory**), 36
  - `FilterCreateArgumentError`, 36
  - `FilterCreateError`, 37
  - `filterDeclOptions()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
35
  - `FilterError`, 37
  - `FilterInfo` (class in **biblamazi.core.bibfilter.factory**),  
37
  - `filtername` (`biblamazi.core.bibfilter.factory.FilterInfo` attribute), 37
  - `filtername()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
36
  - `filterOptions()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
36
  - `FilterOptionsParseError`, 38
  - `FilterOptionsParseErrorHintSInstead`, 38
  - `filterpackagedir`  
(`biblamazi.core.bibfilter.factory.FilterInfo` attribute), 37
  - `filterpackagename`  
(`biblamazi.core.bibfilter.factory.FilterInfo` attribute), 37
  - `filterpackagespec`  
(`biblamazi.core.bibfilter.factory.FilterInfo` attribute), 37
  - `FilterPackageSpec` (class in  
    **biblamazi.core.bibfilter.factory**), 38
  - `filterpath` (`biblamazi.core.bibfilter.factory.FilterInfo` attribute), 38
  - `filterPath()` (`biblamazi.core.biblamazifile.BiblamaziFile` method),  
52
  - `filters()` (`biblamazi.core.biblamazifile.BiblamaziFile` method),  
52
  - `filterVarOptions()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
36
  - `fmodule` (`biblamazi.core.bibfilter.factory.FilterInfo` attribute), 37
  - `fmt()` (`biblamazi.core.bibfilter.factory.FilterCreateArgumentError` method),  
37
  - `fmt()` (`biblamazi.core.bibfilter.factory.FilterCreateError` method),  
37
  - `fmt()` (`biblamazi.core.bibfilter.factory.FilterError` method), 37
  - `fmt()` (`biblamazi.core.bibfilter.factory.FilterOptionsParseError` method),  
38
  - `fmt()` (`biblamazi.core.bibfilter.factory.FilterOptionsParseErrorHintSInstead` method), 38
  - `fname()` (`biblamazi.core.biblamazifile.BiblamaziFile` method),  
52
  - `format()` (`biblamazi.core.blogger.BiblamaziConsoleFormatter` method),  
56
  - `format()` (`biblamazi.core.blogger.ConditionalFormatter` method),  
57
  - `format_filter_help()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
36
  - `format_filter_help()` (in module  
    **biblamazi.core.bibfilter.factory**), 39
  - `formatFilterHelp()`  
(`biblamazi.core.bibfilter.factory.FilterInfo` method), 38
  - `fullFilterPath()` (`biblamazi.core.biblamazifile.BiblamaziFile` method),  
52
- ## G
- `get_action_jobname()` (in module  
    **biblamazi.filters.util.auxfile**), 63
  - `get_all_auxfile_citations()` (in module  
    **biblamazi.filters.util.auxfile**), 63
  - `get_args_parser()` (in module **biblamazi.core.main**),  
59
  - `get_arxiv_cache_access()` (in module  
    **biblamazi.filters.util.arxivutil**), 63
  - `get_copyrihtyear()` (in module  
    **biblamazi.core.butils**), 58
  - `get_github_auth_status()` (in module  
    **biblamazi.core.main**), 59
  - `get_version()` (in module **biblamazi.core.butils**), 58
  - `get_version_split()` (in module  
    **biblamazi.core.butils**), 58
  - `getArgNameFromSOpt()` (`biblamazi.core.bibfilter.factory.DefaultFilterOptions` method),  
36
  - `getArxivApiInfo()` (`biblamazi.filters.util.arxivutil.ArxivFetchedAPIInfoCacheAccessor` method), 61
  - `getArXivInfo()` (`biblamazi.filters.util.arxivutil.ArxivInfoCacheAccessor` method),  
62
  - `getbool()` (in module **biblamazi.core.butils**), 58
  - `getHelpAuthor()` (`biblamazi.core.bibfilter.BibFilter` class method), 32
  - `getHelpDescription()`  
(`biblamazi.core.bibfilter.BibFilter` class method), 32
  - `getHelpText()` (`biblamazi.core.bibfilter.BibFilter` class method), 32
  - `getLoadState()` (`biblamazi.core.biblamazifile.BiblamaziFile` method),  
52
  - `getRunningMessage()`  
(`biblamazi.core.bibfilter.BibFilter` method),

32  
 getSelfLevel() (bibolamazi.core.blogger.BibolamaziLogger method), 57  
 getSOptNameFromArg() (bibolamazi.core.bibfilter.factory.DefaultFilterOptions method), 36  
 GithubAuthSetup (class in bibolamazi.core.argparseactions), 48  
 guess\_encoding\_decode() (in module bibolamazi.core.butils), 58

## H

has\_entry\_for() (bibolamazi.core.bibusercache.tokencheckers.TokenCheckerPerEntry method), 44  
 hasCache() (bibolamazi.core.bibusercache.BibUserCache method), 45  
 helpauthor (bibolamazi.core.bibfilter.BibFilter attribute), 33  
 helpdescription (bibolamazi.core.bibfilter.BibFilter attribute), 33  
 helptext (bibolamazi.core.bibfilter.BibFilter attribute), 33

## I

initFromModuleObject() (bibolamazi.core.bibfilter.factory.FilterInfo static method), 38  
 initialize() (bibolamazi.core.bibusercache.BibUserCacheAccessor method), 47  
 initialize() (bibolamazi.filters.util.arxivutil.ArxivFetchedAPIInfoCacheAccessor method), 61  
 initialize() (bibolamazi.filters.util.arxivutil.ArxivInfoCacheAccessor method), 62  
 insert() (bibolamazi.core.bibusercache.BibUserCacheList method), 48  
 inspect\_getargspec() (in module bibolamazi.core.bibfilter.factory), 40  
 installCacheExpirationChecker() (bibolamazi.core.bibusercache.BibUserCache method), 45  
 instantiateFilter() (bibolamazi.core.bibolamazifile.BibolamaziFile method), 52  
 interactive\_enter() (bibolamazi.core.argparseactions.GithubAuthSetup method), 48  
 interactive\_manage\_token() (bibolamazi.core.argparseactions.GithubAuthSetup method), 48  
 interactive\_setup\_token() (bibolamazi.core.argparseactions.GithubAuthSetup method), 48  
 item\_at() (bibolamazi.core.bibfilter.factory.PrependOrderedDict method), 39

items() (bibolamazi.core.bibusercache.BibUserCacheDic method), 47

## L

latex\_to\_text() (in module bibolamazi.core.butils), 58  
 levelnos (bibolamazi.core.bibfilter.argtypes.LogLevel attribute), 34  
 levelnos\_dict (bibolamazi.core.bibfilter.argtypes.LogLevel attribute), 34  
 levelnos\_list (bibolamazi.core.bibfilter.argtypes.LogLevel attribute), 34  
 load() (bibolamazi.core.bibolamazifile.BibolamaziFile method), 52  
 load\_filterpackage\_providers() (in module bibolamazi.core.main), 59  
 load\_precompiled\_filters() (in module bibolamazi.core.bibfilter.factory), 40  
 loadCache() (bibolamazi.core.bibusercache.BibUserCache method), 46  
 logger (in module bibolamazi.core.blogger), 57  
 LogLevel (class in bibolamazi.core.bibfilter.argtypes), 34  
 longdebug() (bibolamazi.core.blogger.BibolamaziLogger method), 57

## M

main() (in module bibolamazi.core.main), 59  
 make\_filter() (in module bibolamazi.core.bibfilter.factory), 40  
 makeFilter() (bibolamazi.core.bibfilter.factory.FilterInfo method), 38  
 materialize() (bibolamazi.core.bibfilter.factory.FilterPackageSpec method), 39  
 ModuleNotAValidFilter, 39  
 multi\_type\_class() (in module bibolamazi.core.bibfilter.argtypes), 35  
 MultiTypeArgType (class in bibolamazi.core.bibfilter.argtypes), 34

## N

name (bibolamazi.core.bibfilter.factory.FilterInfo attribute), 38  
 name() (bibolamazi.core.bibfilter.BibFilter method), 33  
 new\_token() (bibolamazi.core.bibusercache.tokencheckers.EntryFieldsTokenChecker method), 41  
 new\_token() (bibolamazi.core.bibusercache.tokencheckers.TokenChecker method), 42  
 new\_token() (bibolamazi.core.bibusercache.tokencheckers.TokenCheckerCombine method), 43  
 new\_token() (bibolamazi.core.bibusercache.tokencheckers.TokenCheckerDate method), 43



`new_token()` (bibolamazi.core.bibusercache.  
tokencheckers.TokenCheckerPerEntry  
method), 44

`new_token()` (bibolamazi.core.bibusercache.  
tokencheckers.VersionTokenChecker  
method), 45

`new_value_set()` (bibolamazi.core.bibusercache.  
BibUserCacheDic method),  
47

`NoSuchFilter`, 39

`NoSuchFilterPackage`, 39

`NotBibolamaziFileError`, 56

## O

`opt_action_github_auth` (class in  
bibolamazi.core.argparseactions), 49

`opt_action_help` (class in  
bibolamazi.core.argparseactions), 49

`opt_action_helpwelcome` (class in  
bibolamazi.core.argparseactions), 49

`opt_action_version` (class in  
bibolamazi.core.argparseactions), 49

`opt_init_empty_template` (class in  
bibolamazi.core.argparseactions), 49

`opt_list_filters` (class in  
bibolamazi.core.argparseactions), 49

`opt_set_fine_log_levels` (class in  
bibolamazi.core.argparseactions), 49

`opt_set_verbosity` (class in  
bibolamazi.core.argparseactions), 49

`option_val_repr()` (bibolamazi.core.bibfilter.argtypes.  
EnumArgType method),  
34

`option_val_repr()` (bibolamazi.core.bibfilter.argtypes.  
MultiTypeArgType method),  
34

`option_val_repr()` (bibolamazi.core.bibfilter.argtypes.  
StrEditableArgType method),  
34

`optionSpec()` (bibolamazi.core.bibfilter.factory.  
DefaultFilterOptions method),  
36

`output` (bibolamazi.core.main.ArgsStruct attribute), 59

## P

`package_provider_manager` (in module  
bibolamazi.core.bibfilter.factory), 40

`parse_filterpackage_argstr()` (in module  
bibolamazi.core.bibfilter.factory), 40

`parse_optionstring()` (bibolamazi.core.bibfilter.  
factory.DefaultFilterOptions method),  
36

`parse_optionstring_to_optspec()` (bibolamazi.core.  
bibfilter.factory.DefaultFilterOptions  
method), 36

`parse_timedelta()` (in module bibolamazi.core.butils),  
58

`parseArgdoc()` (in module  
bibolamazi.core.bibfilter.factory), 40

`parseOptionStringArgs()`  
(bibolamazi.core.bibfilter.factory.FilterInfo  
method), 38

`parser()` (bibolamazi.core.bibfilter.factory.  
DefaultFilterOptions method),  
36

`postrun()` (bibolamazi.core.bibfilter.BibFilter method),  
33

`PrependOrderedDict` (class in  
bibolamazi.core.bibfilter.factory), 39

`prerun()` (bibolamazi.core.bibfilter.BibFilter method),  
33

`print_status()` (bibolamazi.core.argparseactions.  
GithubAuthSetup method),  
48

## Q

`quotearg()` (in module bibolamazi.core.butils), 58

## R

`rawConfig()` (bibolamazi.core.bibolamazifile.  
BibolamaziFile method),  
53

`rawHeader()` (bibolamazi.core.bibolamazifile.  
BibolamaziFile method),  
53

`rawRest()` (bibolamazi.core.bibolamazifile.  
BibolamaziFile method),  
53

`rawStartConfigDataLineNo()` (bibolamazi.core.  
bibolamazifile.BibolamaziFile method),  
53

`rebuild_cache()` (bibolamazi.filters.util.arxivutil.  
ArxivInfoCacheAccessor method),  
62

`registerFilterInstance()` (bibolamazi.core.  
bibolamazifile.BibolamaziFile method),  
53

`reloadConfig()` (bibolamazi.core.main.CmdlSettings  
method), 59

`remoteAllowed()` (bibolamazi.core.main.  
CmdlMainPackageProviderManager  
method), 59

`remove_entry_check()` (bibolamazi.core.bibusercache.  
tokencheckers.TokenCheckerPerEntry  
method), 44

`repr()` (bibolamazi.core.bibfilter.factory.  
FilterPackageSpec method),  
39

`requested_cache_accessors()`  
(bibolamazi.core.bibfilter.BibFilter method),  
33

`reset()` (bibolamazi.core.bibolamazifile.  
BibolamaziFile method),  
53

`reset_filters_cache()` (in module  
bibolamazi.core.bibfilter.factory), 40

`resolve_type()` (in module bibolamazi.core.butils), 58

`resolveSourcePath()` (bibolamazi.core.bibolamazifile.  
BibolamaziFile method),  
53

`revalidate()` (bibolamazi.filters.util.arxivutil.  
ArxivInfoCacheAccessor method),  
62

`run_bibolamazi()` (in module bibolamazi.core.main),  
59

run\_biblamazi\_args() (in module  
biblamazi.core.main), 59

runFilter() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
53

## S

save\_github\_auth\_token() (in module  
biblamazi.core.main), 60

save\_token\_and\_exit() (biblamazi.core.  
argparseactions.GithubAuthSetup method),  
48

saveCache() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
53

saveCache() (biblamazi.core.bibusercache.  
BibUserCache method),  
46

saveConfig() (biblamazi.core.main.CmdlSettings  
method), 59

saveRawToFile() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
53

saveToFile() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
54

set\_at() (biblamazi.core.bibfilter.factory.  
PrependOrderedDict method),  
39

set\_items() (biblamazi.core.bibfilter.factory.  
PrependOrderedDict method),  
39

set\_parent() (biblamazi.core.bibusercache.  
BibUserCacheDic method),  
48

set\_time\_valid() (biblamazi.core.bibusercache.  
tokencheckers.TokenCheckerDate method),  
44

set\_validation() (biblamazi.core.bibusercache.  
BibUserCacheDic method),  
48

setBibliographyData() (biblamazi.core.  
biblamazifile.BiblamaziFile method),  
54

setBiblamaziFile()  
(biblamazi.core.bibfilter.BibFilter method),  
34

setCacheObj() (biblamazi.core.bibusercache.  
BibUserCacheAccessor method),  
47

setConfigData() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
54

setDefaultCacheInvalidationTime() (biblamazi.core.  
biblamazifile.BiblamaziFile method),  
54

setDefaultInvalidationTime() (biblamazi.core.  
bibusercache.BibUserCache method),  
46

setEntries() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
54

setInvokationName()

(biblamazi.core.bibfilter.BibFilter method),  
34

setName() (biblamazi.core.bibfilter.factory.FilterError  
method), 37

setRawConfig() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
55

setShowPosInfoLevel() (biblamazi.core.blogger.  
BiblamaziConsoleFormatter method),  
56

setup\_and\_get\_arxiv\_accessor() (in module  
biblamazi.filters.util.arxivutil), 63

setup\_filterpackage\_from\_argstr() (in module  
biblamazi.core.main), 60

setup\_filterpackages\_from\_env() (in module  
biblamazi.core.main), 60

setup\_from\_arg() (biblamazi.core.argparseactions.  
GithubAuthSetup method),  
48

setup\_simple\_console\_logging() (in module  
biblamazi.core.blogger), 57

sourceLists() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
55

sources() (biblamazi.core.biblamazifile.  
BiblamaziFile method),  
55

store\_key\_bool (class in  
biblamazi.core.argparseactions), 49

store\_key\_const (class in  
biblamazi.core.argparseactions), 49

store\_key\_val (class in  
biblamazi.core.argparseactions), 49

store\_or\_count (class in  
biblamazi.core.argparseactions), 49

StrEditableArgType (class in  
biblamazi.core.bibfilter.argtypes), 34

stripArXivInfoInNote() (in module  
biblamazi.filters.util.arxivutil), 63

## T

token\_for() (biblamazi.core.bibusercache.  
BibUserCacheDic method),  
48

TokenChecker (class in biblamazi.core.bibusercache.  
tokencheckers),  
42

TokenCheckerCombine (class in biblamazi.core.  
bibusercache.tokencheckers),  
42

TokenCheckerDate (class in biblamazi.core.  
bibusercache.tokencheckers),  
43

TokenCheckerPerEntry (class in biblamazi.core.  
bibusercache.tokencheckers),  
44

type\_arg\_input (biblamazi.core.bibfilter.argtypes.  
ColonCommaStrDict attribute),  
34

type\_arg\_input (biblamazi.core.bibfilter.argtypes.  
CommaStrList attribute),  
34

type\_arg\_input

(bibolamazi.core.bibfilter.argtypes.LogLevel attribute), [34](#)

## U

unset\_token\_and\_exit() (bibolamazi.core.  
argparseactions.GithubAuthSetup method),  
[49](#)

use\_auto\_case() (bibolamazi.core.bibfilter.factory.  
DefaultFilterOptions method),  
[36](#)

use\_cache (bibolamazi.core.main.ArgsStruct attribute),  
[59](#)

## V

validate() (bibolamazi.core.bibusercache.  
BibUserCacheDic method),  
[48](#)

validate\_filter\_package() (in module  
bibolamazi.core.bibfilter.factory), [40](#)

validate\_item() (bibolamazi.core.bibusercache.  
BibUserCacheDic method),  
[48](#)

validateOptionStringArgs()  
(bibolamazi.core.bibfilter.factory.FilterInfo  
method), [38](#)

verbosity\_logger\_level() (in module  
bibolamazi.core.main), [60](#)

version\_str (in module bibolamazi.core.version), [60](#)

VersionTokenChecker (class in bibolamazi.core.  
bibusercache.tokencheckers),  
[44](#)

## W

warn\_deprecated() (in module bibolamazi.core.butils),  
[59](#)