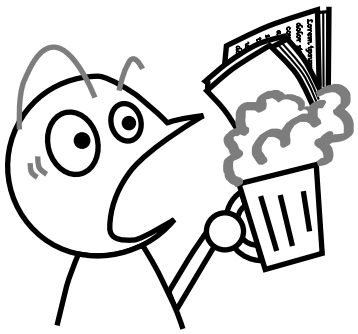


BIBULOUS

A drop-in BibTeX replacement based on style templates.

Bibulous Documentation

Release 1.0

Bibulous developers

July 09, 2013

CONTENTS

Contents:

CHAPTER

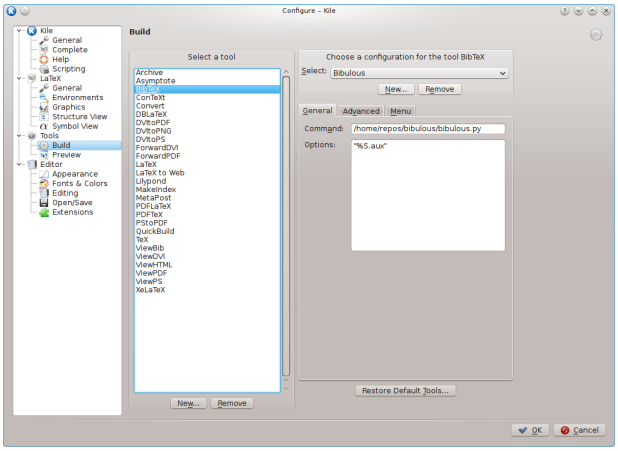
ONE

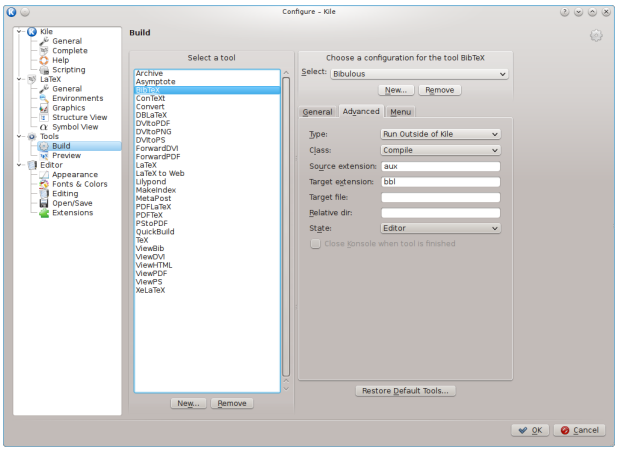
GETTING STARTED

For general users, all that is needed is place the main `bibulous.py` file into the Python path. For users interested in using the auxiliary commands, `bibulous_authorextract.py` and `bibulous_citeextract.py` must also be in the Python path, and must be in the same directory as the main file.

1.1 Kile: replacing BibTeX with Bibulous

1. In your `.tex` file, change the filename of the `\bibliography{...}` command to the filename for the appropriate Bibulous-format bibliography style template (`.bst` file).
2. In Kile, go to the menu bar and select `Settings` > `Configure Kile`. Select `Tools` > `Build` and choose `BibTeX` from the `Select a tool` menu (see the figure). To the right of the menu, after you select `BibTeX` you should see "Choose a configuration for the tool BibTeX". Below the drop-down menu, select the button "New" and type in the name `Bibulous` (or whatever you prefer to call your new tool). Below, in the `General` tab, type in the location of the `bibulous.py` file. And in the `Options` field, type `%dir_base/%S.aux`.





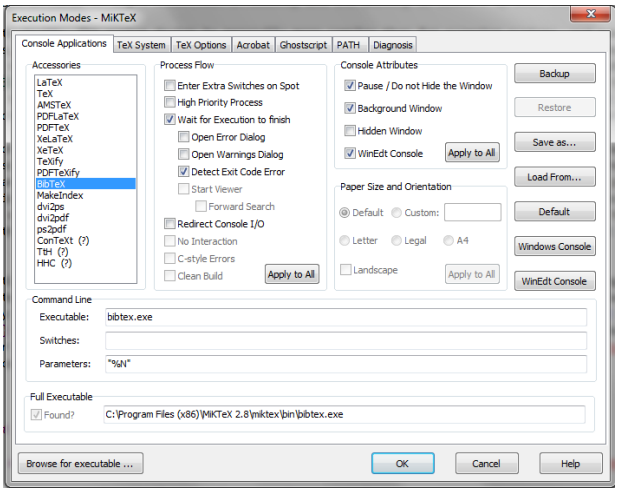
That should be it. In case your default setup is different, you can also check the `Advanced` tab settings and verify that they are as shown in the second figure. (That is, `Source extension` is set to `aux`, and `Target extension` is set to `bbl`.)

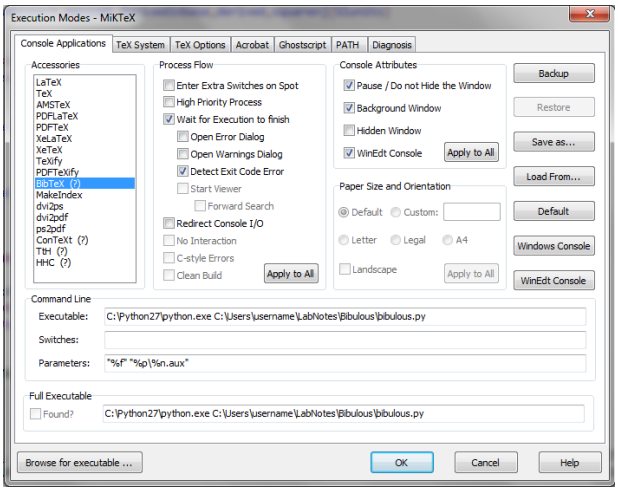
3. Note that the following variables are accessible in Kile's `Options` field:

```
%source = filename (i.e. filename with suffix but not path)
%S = filename without suffix (and without path)
%dir_base = source file directory (source file's path without the filename)
%dir_target = target file directory (source file's path without the filename)
```

1.2 Modifying WinEdt5 to replace BibTeX with Bibulous

1. Go to the menu `Options` > `Execution Modes`. In the `Console Applications` menu on the left hand side, select `BibTeX`. Then replace the three `Command Line` fields with the ones shown in the figure, replacing the files paths with the ones correct for your installation of Python and `bibulous.py`.





2. Note that the following are definitions of WinEdt registers:

```
%f = full path of active file (= %p/%n.%t)
%n = name of the active file
%p = the path of the active file
%t = the extension of the active file
%q = the path relative to the main file (i.e. for subdirectories)
%b = WinEdt's local working directory (not the tex file directory)
%B = path to the WinEdt executable file
```

CHAPTER

TWO

GUIDELINES FOR WRITING BIBLIOGRAPHY STYLE TEMPLATES

1. Comments begin with #. (A single # indicates a symbol and not a comment!)
2. A line which begins with an entrytype name followed by an = sign defines the bibliographic format for that entrytype. For example:

   ```
   article = <authorlist>, ``<title>,'' <journal>, <volume>, ...
             [<startpage>--<endpage>|<eid>|]  (<year>). [<note>.]
   ```

   Here the `article` entrytype will be typeset so that the list of authors is followed by the article title in double quotes, the journal name in standard font (i.e. not italics), the volume number, the page range, and the year.
3. Square brackets `[]` indicate an optional entry; any required entries which are not defined in the BibTeX database file (.bib file) are replaced with '???' by default. Optional arguments is undefined are simply skipped. If a `|` is present within the square brackets, it indicates an "elseif" argument (if not the final `|` within the brackets), or an "else" argument (if the final `|`). The "else" indicates a *required* argument, so if you want an optional entry to be replaced with something, you can use `[option|]` — the use of an empty cell inside the square brackets indicates that we simply use the default replacement for an undefined required argument (i.e. `???`). If you want all of the cells to be optional, then use `|"]` in the last cell – that is, the last cell should be an empty string. For now, the format currently does not allow nesting of brackets.
4. If you need the square brackets or `|` symbol as formatting elements, then simply use `{\makeopenbracket}`, `{\makeclosebracket}`, or `{\makeverticalbar}`. If you need the angle brackets as formatting elements, then use `{\makegreaterthan}` and `{\makelessthan}`. Note that the curly brackets are needed here so that when Bibulous replaces the command with the appropriate symbol, that symbol can be used correctly in LaTeX commands.
5. Unlike BibTeX, Bibulous does *not* change the capitalization state of any entry variables. It assumes that the authors have defined it the way they want it.
6. If an entrytype format definition contains only another entrytype on the right hand side of the =, for example:

   ```
   inbook = incollection
   ```

   then this simply defines the format for the entrytype on the left hand side as identical to that of the entrytype given on the right hand side.
7. The second type of data present in the file are the formatting options. These are defined by writing `options` followed by a period and then the option name, for example:

   ```
   options.authorlist_format = 'first_name_first'
   ```

8. Bibulous only allows string variables to be inserted into a given position within an entrytype template, and all Bibulous variables are surrounded with angle brackets. Thus, when typesetting the bibliography, Bibulous will

replace the variable <authorlist> with the string stored in the authorlist field of the current entry being formatted. An example list of variables one may choose to use is:

<authorlist>, <booktitle>, <chapter>, <edition_ordinal>, <editorlist>, <eid>, <endpage>, <institution>, <journal>, <nationality>, <note>, <number>, <organization>, <publisher>, <school>, <series>, <startpage>, <title>, <version>, <volume>, <year>.

As one can see, these primarily consist of the various fields one can expect to see within a given BibTeX-formatted database file. This list is actually freely extensible. You can add whatever additional variables you like, so that if you use a special video field in your database, you can insert that field's value into the template wherever <movie> is located.

9. Note that several fields are defined by default which are *not* directly from the bibliography database. These are authorlist, editorlist, startpage, endpage, and edition_ordinal. These fields are derived from the original database file, but have been reformatted.

10. Although the entrytype template definitions listed below are in alphabetical order, that can be put in any desired order within the file. (The exception to this rule is that if a definition consists of, for example:

```
inbook = incollection
```

then the incollection template must already be defined. Also note that two entrytype names are special and so cannot be used on the left hand side of the equals sign here: comment and preamble.

11. Options for citation_order include

- citenumber or none (the default)
- citekey
- alpha (uses three letters of author's last name plus last two numbers in the year)
- nyt or plain (uses the first author's last name, the year, and then the title)
- nty, nyvt, anyt, anyvt, ynt, and ydnt

where the different letters indicate (as in Biblatex) n = name (i.e. author's last name), y = year, t = title, v = volume, and a = alphabetic label (where the user is implementing some bibliography front-end that prints out alphabetic labels inside the .aux file). The d here means that the order should be *descending* rather than the default of ascending.

12. A user wanting a localized form of quotation should use \enquote{<title>} rather than ``<title>'', and add \usepackage{csquotes} to the preamble of the LaTeX document.

CHAPTER

THREE

NOT YET DOCUMENTED

1. Need to give instructions to users about how to overwrite the `generate_sortkey()` function for arbitrary citation sorting.
2. Also need instructions on how to overwrite the `generate_bibitem_label()` function for generating the reference item labels.

CHAPTER

FOUR

INSTRUCTIONS ON HOW TO REPORT A BUG TO THE BIBULOUS DEVELOPMENT TEAM

4.1 Where to report a bug

Send an email to the `users_mailing_list`. Once it's confirmed as a bug, someone, possibly you, can enter it into the issue tracker. (Or if you're pretty sure about the bug, go ahead and post directly to the developers mailing list, `developers_mailing_list`. But if you're not sure, it's better to post to [`users mailing list`] first; someone there can tell you whether the behavior you encountered is expected or not.)

4.2 How to report a bug

First, make sure it's a bug. If Bibulous does not behave the way you expect, look in the documentation and mailing list archives for evidence that it should behave the way you expect. If the documentation and archives do not contain enough information to tell you whether the behavior is a bug or is expected behavior, go ahead and ask on the users mailing list first `users_mailing_list`. Also check that you are running the most recent version of Bibulous. It may be that the bug has already been fixed.

Once you've established that it's a bug, the most important thing you can do is come up with a simple description and reproduction recipe. For example, if the bug, as you initially found it, involves five files over ten commits, try to make it happen with just one file and one commit. The simpler the reproduction recipe, the more likely a developer is to successfully reproduce the bug and fix it.

When you write up the reproduction recipe, don't just write a prose description of what you did to make the bug happen. Instead, give a copy of the exact series of commands you ran, and their output. Use cut-and-paste to do this. If there are files involved, be sure to include the names of the files, and even their content if you think it might be relevant. The very best thing is to package your reproduction recipe as a script, that helps a lot.

In addition to the reproduction recipe, we'll also need a complete description of the environment in which you reproduced the bug. That means:

- Your operating system
- The Python version you are running under.
- The release and/or revision of Bibulous.
- Anything else that could possibly be relevant. Err on the side of too much information, rather than too little.

Once you have all this, you're ready to write the report. Start out with a clear description of what the bug is. That is, say how you expected Bibulous to behave, and contrast that with how it actually behaved. While the bug may seem obvious to you, it may not be so obvious to someone else, so it's best to avoid a guessing game. Follow that with the environment description, and the reproduction recipe. If you also want to include speculation as to the cause, and even suggest how the code may be modified to fix the bug, that's great.

Post all of this information to the developers mailing list, `developers_mailing_list`, or if you have already been there and been asked to file an issue, then go to the Issue Tracker and follow the instructions there.

Thanks! It's a lot of work to file an effective bug report, but a good report can save hours of a developer's time, and make the bug much more likely to get fixed.

CHAPTER

FIVE

DEVELOPER GUIDE

5.1 Guidelines for Python coding style

1. Note that you can mix 8-bit Python strings (ASCII text) with UTF-8 encoded text as long as the 8-bit string contains only ASCII characters.
2. Keep in mind when running into Unicode errors: reading a line of text from a file produces a line of bytes and not characters. To decode the bytes into a string of characters, you need to know the encoding.
3. There are a couple of minor points where the Bibulous coding standards deviates from Python's PEP8:
   (a) A line width of 100 is the standard (not 80).
   (b) In general, statements that evaluate to a boolean are placed within parentheses (i.e. `if (a < b):` rather than `if a < b:`).
4. Many developers prefer to spread out code among a large number of small files. Bibulous is currently organized in the opposite fashion – all of the code needed to run `bibulous` to create a `.bbl` file is located within a single large file. Several auxiliary scripts exist, but these use `bibulous.py` as a core library file, and perform different tasks (such as extracting sub-bibliography databases) than the main file was designed to do.

5.2 Overall project strategy and code structure

The basic function of BibTeX is to accept an `.aux` file as input and to produce a `.bbl` file as output. The `aux` file contains all of the citation information as well as the filenames for the bibliography database file (`.bib`) and the style file (`.bst`).

The basic program flow is as follows:

1. Read the .aux file and get the names of the bibliography databases (.bib files), the style templates (.bst files) to use, and the entire set of citations.
2. Read in all of the bibliography database files into one long dictionary (*bibdata*), replacing any abbreviations with their full form. Cross-referenced data is *not* yet inserted at this point. That is delayed until the time of writing the BBL file in order to speed up parsing.
3. Read in the Bibulous style template file as a dictionary (*bstdict*).
4. Now that all the information is collected, go through each citation key, find the corresponding entry key in *bibdata*. If there is crossref data, then fill in missing values here. Also create the "special fields" here. Finally, from the entry type, select a template from *bstdict* and begin inserting the variables one-by-one into the template.

Because the `.bib` file is highly structured, it is straightforward to write a parser by hand in Python: the `parse_bibfile()` method converts the `.bib` file contents into a Python dictionary (the `Bibdata` class'

bibdata). The .aux file is even easier to parse, and the parse_auxfile() method converts the citation information into the Bibdata class' citedict dictionary. The .bst style template file, having its own domain specific language, is much more complicated, so that its parser is generated from a grammar written for the Antlr parser generator. (This creates Bibulous' only external dependency – Java – which we may be able to eliminate if we use a Python-based parser generator, such as pyparsing.)

The Bibdata class thus holds all relevant information needed to operate on a bibliography and generate the output LaTeX-formatted .bbl file.

5.3 Parsing BIB files

5.3.1 parse_bibfile()

The strategy for parse_bibfile() is to find each individual bibliography entry, determine its entry type, and save all of the text between the entry's opening and closing braces as one long string, to be passed to parse_bibentry() for parsing. To gather the entry data string, we first look for the next line that starts with @. On that line, we look for a string after the @ followed by {, where the string gives the entry type. After we know the entry type, we look for the corresponding closing brace. If we don't find it on the same line, then we read in the next line, and so forth, concatenating all of the lines into one long "entry string" until we encounter the corresponding closing brace. Once we have this extended "entry string" we feed it to parse_bibentry() to generate the bibliography data. Once we have come to the end of a given entry, we continue reading down the file looking for the next '@' and so on.

Although this approach effectively means that we have to pass twice through the same data, dealing with brace-matching can otherwise become a mess since the BibTeX format, since it allows nested delimiters, is not directly compatible with regular expressions.

5.3.2 parse_bibentry()

parse_bibentry() only needs to worry about a single entry, and there are four possible formats for the entry string passed to the function:

1. If the entrytype is a comment, then skip everything, adding nothing to the database dictionary.
2. If the entrytype is a preamble, then treat the entire entry contents as a single fieldvalue. Append the string onto the preamble value in the bibdata dictionary.
3. If the entrytype is a string (i.e. an abbreviation), then there is no entrykey. Get the fieldname (abbreviation key), and the remainder of the string is a single field value (the full form of the abbreviated string. Add this key-value pair to the abbrevs dictionary.
4. If the entry is any other type, then get the entrykey, and the remainder of the string is a *series* of field-value pairs.

Once it determines which of these four options to use, parse_bibentry() extracts the entry key (if present), separates out each of the fields (if more than one is present) and loops over each field with a call to parse_bibfield() to extract the field key-value pairs.

5.3.3 parse_bibfield()

parse_bibfield() is the workhorse function of the BIB parsing. And because of BibTeX's method for allowing concatenation, use of abbreviation keys, and use of two different types of delimiters ("...") or {...}), this function is a little messy. However, for the format of a given field, there are four parsing possibilities:

1. If the field begins with a double quote " then scan until you find the next ". Add that to the result string. If the ending " is followed by a comma, then the field is done; return the result string. If the ending is followed by a # then expect another field string. Scan for it and append it to the current result string.
2. If the field begins with { then scan until you resolve the brace level. This should be followed by a comma, since no concatenation is allowed of brace-delimited fields. Otherwise issue a syntax error warning.
3. If the field begins with a # (concatenation operator) then skip whitespace to the next character set, where you should expect a quote-delimited field. Append that to the current result string.
4. If the field begins with anything else, then the substring up until the first whitespace character represents an abbreviation key. Locate it and substitute it in. If you don't find the key in the `abbrevs` dictionary, give a warning and skip.

5.4 Parsing AUX files

The `.aux` file contains the filenames of the `.bib` database file and the `.bst` style template file, as well as the citations. The `get_bibfilenames()` method scans through the `.aux` file and locates a line with `\bibdata{...}` which contains a filename or a comma-delimited list of filenames, giving the database files. Another line with `\bibstyle{...}` gives the filename or comma-delimited list of filenames for style templates. The filenames obtained are saved into the `filedict` attribute – a dictionary whose keys are the file extensions `aux`, `bbl`, `bib`, `bst`, or `tex`.

The `parse_auxfile()` method makes a second pass through the `.aux` file, this time looking for the citation information. (Auxiliary files are generally quite small, so taking multiple passes through them cost very little time.) Each line with `\citation{...}` contains a citation key or comma-delimited list of citation keys – each one is added into the citation dictionary (`citedict`), with a value corresponding to the citation order.

5.5 Parsing BST files

(This part is changing at the moment, and so the documentation is not available yet.)

Note format_bibitem() is where we compile any scripts present in the BST files. Doing it before this step basically requires that we run any script on all of the entries of the database and not just the ones that have been cited. For large databases, this can be a significant amount of extra computation. There are two different compilation steps here. The first is that, before looping over the bibitems, we compile the scripts so that any functions defined there are available to the local namespace. Second, once we're inside the loop, if the template string for the current entry has a variable which is a user-defined variable, then we have to use Python's `eval()` function to obtain the result of evaluating the script on the current entry.

5.6 Writing the BBL file

Now that all the information is available to Bibulous, we can begin writing the output BBL file. First we write a few lines to the preamble, including the `preamble` string obtained from the `.bib` database files. We also create the citation list – the citations listed in the sorting order as defined in the style template files. (This requires a surprising amount of code to get right – see **Generating sortkeys** below.) We loop over each citation in the desired order, and insert cross-reference information to fill in missing fields, and parse each name field (see the "Formatting names" subsection below). The cross-referencing and name parsing steps can be delayed until later on in the processing chain, but would require more complex code to do there, so doing them here keeps the code simpler without sacrificing much speed. (The assumption here is that the citation list is small, at least in comparison to the database, so that limiting the difficult parsing to only those entries cited will allow significant improvement in speed.) Finally, at each

step in the loop, we call `format_bibitem()` to insert the database entry fields into the appropriate style template, incorporating any extra formatting requested by the user in the style template file.

5.7 Name formatting

One of the more complex tasks needed for parsing BIB files is to resolve the elements of name lists (typically saved in the `author` and `editor` fields). In order to know how these should be inserted into a template, it is necessary to know which parts of a given person's name correspond to the first name, the middle name(s), the "prefix" (or "von part"), the last name (or "surname"), and the "suffix" (such as "Jr." or "III"). These five pieces or each person's name are saved as a dictionary, so that a bibliography entry with five authors is represented in `<authorlist>` as a list of five dictionaries, and each dictionary having keys `first`, `middle`, `prefix`, `last`, and `suffix`.

In order to speed up parsing times, the actual mapping of the `author` or `editor` fields to `authorlist` or `editorlist` is not done until the loop over citation keys performed while writing out the BBL file. The function that product the list-of-dicts parsing result is `namestr_to_namedict(namestr)`.

The default formatting of a namelist into a string to be inserted into the template is performed by `format_namelist()`.

5.7.1 create_namelist()

A BibTeX "name" field can consist of three different formats of names:

1. A space-separated list: `[firstname middlenames suffix lastname]`
2. A two-element comma-separated list: `[prefix lastname, firstname middlenames]`
3. A three-element comma-separated list: `[prefix lastname, suffix, firstname middlenames]`

So, an easy way to separate these three categories is by counting the number of commas that appear. The trickiest part here is that although we can use `and` as a name separator, we are only allowed to do so if `and` occurs at the top brace level.

In addition, in order to make name parsing more flexible for nonstandard names, Bibulous adds two more name formats to this list:

4. A four-element comma-separated list: `[firstname, middlenames, prefix, lastname]`
5. A five-element comma-separated list: `[firstname, middlenames, prefix, lastname, suffix]`

For each name in the field, we parse the name tokens into a dictionary. We then compile all of the dictionaries into a list, ordered by the appearance of the names in the input field.

5.7.2 format_namelist()

Given a namelist (list of dictionaries), we glue the name elements together into a single string, incorporating all of the format options selected by the user in the template file. This includes calls to `namedict_to_formatted_namestr()`, and to `initialize_name()` if converting any name tokens to initials.

5.8 Generating sortkeys

If the user's style template file selects the citation order to be `citenum` or `none`, then creating the ordered citation list is as simple as listing the citation keys in order of their citation appearance, which was recorded as the value in the citation dictionary. If the user instead chooses the citation order to be `citekey`, then all that is needed is to sort the citation keys alphabetically. Similar operations follow for the various citation order options, but the difficult lies in correctly sorting in the presence of non-ASCII languages, and especially in the presence of LaTeX markup of non-ASCII names. For a citation sorting order that requires using author names, any LaTeX markup needs to be converted to its Unicode equivalent prior to sorting. Using unicode allows the sorting to be done with any input languages, and allows the sorting order to be locale-dependent.

`create_citation_list()` is the highest-level function for generating the citation list. For each citation key, it calls `generate_sortkey()`, which is the workhorse function for including all of the various options when generating the key to use for sorting the list. A key part of the function is a call to `purify_string()`, which removes unnecessary LaTeX markup elements and then calls `latex_to_utf8()` to convert LaTeX-markup non-ASCII characters to Unicode. It is only after all of these conversions that the final sorting is performed and the sorted citation list returned.

5.9 Testing

The suite of regression tests for Bibulous consist of various template definitions and database entries designed to test individual features of the program. The basic approach of the tests is as follows:

1. Once a change is made to the code (to fix a bug or add functionality), the developer also adds an entry to the `test/test1.bib` file, where the entry's "entrytype" is named in such a way to give an indication of what the test is for. For example, the entry in the BIB file may be defined with:

   ```
   @test_initialize1{...
   ```

   where the developer provides an `author` field in the entry where one or more authors have names which are difficult to for generating initials correctly. The developer should also include at least a 1-line comment about the purpose of the entry as well. To make everything easy to find, use the entrytype as the entry's key as well. Thus, the example above would use:

   ```
   @test_initialize1{test_initialize1, ...
   ```

2. If the above new entry is something which can be checked with normal options settings, then the developer should add a corresponding line in the BST file defining how that new entrytype (i.e. `test_initialize1`) should be formatted. If *different* options settings are needed, then a new BST file is needed. Only a minimalist file is generally needed: the file can, for example, contain one line defining a new entrytype and one line to define the new option setting. You can define all of the other options if you want, but these are redundant and introduce a number of unnecessary "overwriting option value..." warning messages.

3. Next, the developer should add a line `\citation{entrytype}` to the AUX file where the key is the key given in the new entry of the BIB file you just put in (e.g. `test_initialize1`). This is the same as the entrytype to keep everything consistent.

4. Next, the developer needs to add two lines to the `test1_target.bbl` file to say what the formatted result should look like. Take a look at other lines to get a feel for how these should look, and take in consideration the form of the template just added to the BST file.

5. Finally, run `bibulous_test.py` to check the result. This script will load the modified BIB and BST files and will write out several formatted BBL file `test1.bbl` etc. It will then run a `diff` program on the output file versus the target BBL file to see if there are any differences between the target and actual output BBL files.

5.10 Generating the documentation

From the bibulous repository `doc/` subfolder, run `make html` to generate the HTML documentation. The result can be found in `doc/_build/html/`, with `index.html` as the main file. To generate the PDF documentation, run `make latexpdf` from the `doc/` subfolder, with the result found at `doc/_build/latex/Bibulous.pdf`.

CHAPTER

SIX

OVERVIEW

Bibulous is a drop-in replacement for BibTeX that makes use of style templates instead of BibTeX's BST language. The code is written in Python and, like BibTeX itself, is open source.

Bibulous developed out of frustration with the complexity of creating bibliography styles using BibTeX's obscure language, and also from the realization that because bibliographies are highly structured, one should be able to specify them simply and flexibly using a template approach. There should be no need to learn a new language just to build a bibliography style, and specifying a style should taken only a matter of minutes.

Bibulous incorporates this template approach, and at the same time implements many of the modern enhancements to BibTeX, such as the ability to work with languages other than English, better support for allowing non-standard bibliography entry types, functionality for enhanced citation styles, and increased options for author name formatting, among others. The fact that Bibulous is agnostic to the names of fields in bibliography database files (`.bib`) means that one can use the same database and same LaTeX commands for generating each of: a bibliography, a glossary, a publications list for a CV, an annotated bibliography, and more, all within the same file and only by specifying a different style template for each case.

Bibulous' "style template" files allow a user to visualize the entire bibliography format structure in a concise way within a single page of text. Moreover, the template is structured with its own Python-like mini-language, intended to allow uses to create flexible formatting instructions quickly and easily. The example below illustrates the simplicity of the format.

6.1 Example

For a very simple bibliography, consisting of only journal articles and books, a complete style template file may consist of just two lines::

```
article = <authorlist>, \enquote{<title>,} \textit{<journal>} \textbf{<volume>}: ...
              [<startpage>--<endpage>|<startpage>|<eid>|] (<year>).[ <note>]
book = [<authorlist>|<editorlist>|], \textit{<title>} (<publisher>, <year>)...
              [, pp.~<startpage>--<endpage>].[ <note>]
```

The `<variable>` notation indicates that the corresponding bibliography entry's field is to be inserted into the template there. The `[...|...]` notation behaves similar to an if...elseif statement, checking whether a given field is defined within the bibliography entry. If not defined, then it attempts to implement the instruction in the block following the next | character.

We can read the above article template as indicating the following structure for LaTeX-formatting the cited entry in the bibliography (`.bib` file). For articles, we first insert the list of author names (formatted according to the default form), followed by a comma. If no `author` field was found in the bibliography entry, then insert `???` to indicate a missing required field. Next insert a quoted title, followed by an italicized journal name, and a boldface volume number (all of these are required fields). Next, if the `pages` field was found in the entry's database, then parse the start and end page

numbers and insert them here. If the `pages` field indicates that there was only one page, then use that instead. Or if the `pages` field is not present, then check to see if the `eid` is defined, and use it instead. However, if none of these three possibilities are available, then insert the "missing field" indicator, `???`. Finally, put the year inside parentheses, and if the `note` field is defined in the entry, then add that to the end (following the period). If `note` is not defined, then just ignore it.

One can read the book template similarly, and find that it has different required and optional fields. The simplicity of the format allows one to customize databases to suit any use. For example, to use a bibliography entrytype `X` instead of `book`, then all that is necessary is to go into the template file and change `book` to `X`. Of, if you wish to add a new field, such as translator, then if it has been added to the `.bib` database file, one need only add some text to the template, say `(: <translator>)` to insert that into every bibliography entry which has translator defined for that entrytype.

6.2 Installing and instructions

Instructions for installing Bibulous, and for seamlessly integrating it into your normal LaTeX workflow, are given in the `INSTALL.rst` file. Users can also consult the user guide (`user_guide.rst`) for further information and tutorials. A `FAQ` page is also available.

6.3 Developers

Bibulous is a brand new project, and so it has so far been a solo effort. Anyone interested in helping out is welcome to join; just send an email to the developers mailing list and we will try to help you get involved and show you the ropes. And, this being the maintainer's first open source project, any suggestions by experienced developers are welcome.

Guidelines for developers are given in `developer_guide.rst`, and includes an overview of the project's strategy and overall code structure. Note that a bug tracking system has not yet been set up for the project. HTML-based documentation is provided in `/bibulous/doc/_build/html/index.html`, and a corresponding PDF file in `/home/repos/bibulous/doc/_build/latex/Bibulous.pdf`. The `setup.py` and `MANIFEST.in` files provided in the repository base directory are used to create a Python package using the `disutils` distribution utilities module.

6.4 License

Bibulous is released under the MIT/X11 license, meaning that it is free and open source, and that it can be used without restriction in other programs, commercial or not. The license is given in the file `LICENSE.txt`, the text of which is reproduced here:

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *search*