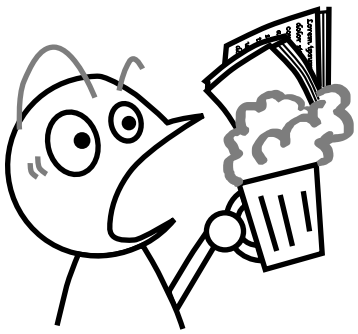


BIBULOUS

A drop-in BibTeX replacement based on style templates.

Bibulous Documentation

Release 1.2

Bibulous developers

December 13, 2013

CONTENTS

Contents:

CHAPTER

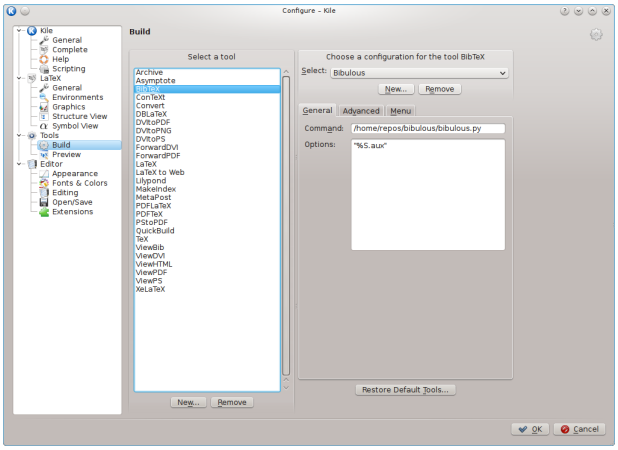
ONE

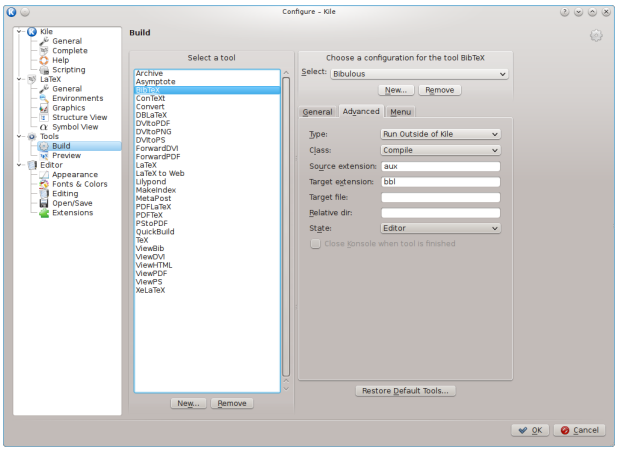
GETTING STARTED

For general users, all that is needed is place the main `bibulous.py` file into the Python path and to tell LaTeX to use Bibulous rather than BibTeX as their bibliography engine. For users interested in using the auxiliary command `bibulous_authorextract.py`, this file must also be in the Python path, and must be in the same directory as the main file. It is also possible to tell any LaTeX front-end to use Bibulous in place of BibTeX. For Kile (on Linux) and WinEdt (on Windows), instructions for doing this are given below.

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`.





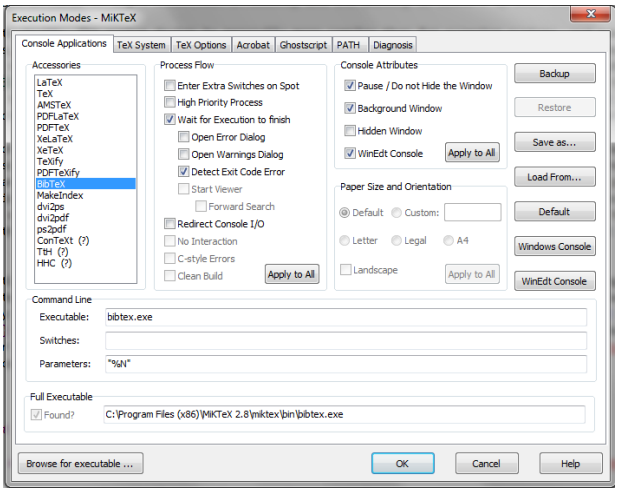
That should be it. In case your default setup is different, you can also check the `Advanced` tab settings and verify that they are as shown in the second figure. (That is, `Source extension` is set to `aux`, and `Target extension` is set to `bbl`.)

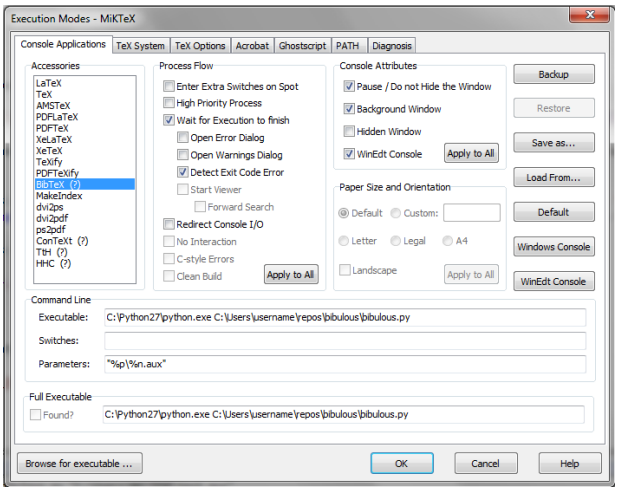
3. Note that the following variables are accessible in Kile's `Options` field:

```
%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

2.1 Syntax

1. Comments begin with `#`, following the Python convention.
2. Each template file can have as many as five sections. None of the sections are required to be in the file, but any definitions in the file must be placed inside a section header so that the code knows how to deal with the definition. The four possible section headers are: TEMPLATES, SPECIAL-TEMPLATES, OPTIONS, VARIABLES, DEFINITIONS. And note that a section header is always placed by itself on a line and has a colon appended to it, as in `TEMPLATES:`.
3. The `TEMPLATES` section of the file contains template definitions for formatting references. The `SPECIAL-TEMPLATES` section contains definitions for creating variables within each database entry. The `OPTIONS` section contains definitions for various keywords that can be used to modify program behavior. The `VARIABLES` section contains user definitions for new variables to be made available. The difference between these definitions and those in `SPECIAL-TEMPLATES` is that the ones provided in the `VARIABLES` section are in-line Python code, whereas the former use templates. Finally, the `DEFINITIONS` section of the file contains Python-executable code that can then make functionality available in the form of template variables. (An example is provided in the *Python API* section below.
4. All variable definitions within the TEMPLATES, SPECIAL-TEMPLATES, and OPTIONS sections use the variable name followed by whitespace, an equals sign, more whitespace, and then the definition itself. Thus the [whitespace+=+whitespace] expression is the delimiter between variable and definition, and is required syntax.
5. An ellipsis `...` at the end of a line indicates a line continuation. All whitespace following the ellipsis, and all whitespace preceding text on the next line, is removed from the resulting connected text.
6. An ellipsis in the middle of a line indicates an "implicit loop". For details, see the *Examples for namelist formatting* section below.
7. Inside the `TEMPLATES` section, all of the variable definitions are intended to map to a database entrytype name. For example:

   ```
   article = <au>, ``<title>,'' <journal>, <volume>, <startpage>--<endpage> (<year>).
   ```

   Here the `article` entrytype will be typeset so that the list of authors (`<au>`) 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.
8. A variable is indicated by angle brackets, as `<var>` and represents the contents of a field found within the bibliography database. 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 typical variables one might use is:

`<au>`, `<booktitle>`, `<chapter>`, `<edition>`, `<ed>`, `<eid>`, `<endpage>`, `<institution>`, `<journal>`, `<nationality>`, `<note>`, `<number>`, `<organization>`, `<publisher>`, `<school>`, `<series>`, `<startpage>`, `<title>`, `<version>`, `<volume>`, `<year>`.

This list is actually freely extensible. A user can add any additional variables needed, so that if a `video` field is used in a `.bib` database file, then this can be used within a formatted reference simply by placing `<video>` into the template wherever the information needs to be inserted.

9. Any variable placed within square brackets `[]` indicate that it is an optional variable — it is not required that the database have that entry. While required entries that are not defined in the BibTeX database file (.bib file) are replaced with '???', and undefined optional variables are simply skipped. If a `|` is present within the square brackets, it indicates an "elseif" clause. That is, if the template is `[<var1>|<var2> and <var3>]`, then the code will look for `var1` as a field within the current database entry being formatted. If it does not find the entry, then it will try the next block, where it finds the two variables `var2` and `var3`. If both are defined, then the original template `[<var1>|<var2> and <var3>]` is replaced with `<var2> and <var3>` (i.e. removing the square brackets) and proceeds to replace the two variables with their corresponding fields. If either one of `var2` or `var3` is undefined in the entry, then the entire optional [...] portion of the template is skipped.

10. If the `|` symbol is used to create an empty last cell, as in `[<var1>|<var2>|]`, this indicates that while the individual cells within the optional block are themselves optional, it is required to have at least _one_ among the cells to be defined. Thus, `[<note>|]` has the same meaning as simply `<note>` does.

11. Nesting of `[]` brackets is allowed, but the syntax becomes computationally expensive to parse, so that these structures should be used sparingly:

12. Users that need to use `[`, `]`, `#`, `<`, `>`, or `|` symbols as formatting elements within the reference list can implement them using some custom LaTeX-markup commands: `{\makeopenbracket}`, `{\makehashsign}`, `{\makeclosebracket}`, `{\makegreaterthan}`, `{\makelessthan}`, or `{\makeverticalbar}`. Note that the curly brackets used in each case are required.

13. In the `TEMPLATES` section of the file, if an entrytype format definition contains only another entrytype name on the right hand side of the =, i.e.:

    ```
    inbook = incollection
    ```

    then this means that the existing `incollection` template should be copied for use with `inbook` entrytypes. (Note that `incollection` should be defined above this line in order for this to work.)

14. Note that several fields are defined by default which are *not* directly from the bibliography database. These are `au`, `authorlist`, `citekey`, `citenum`, `ed`, `editorlist`, `endpage`, and `startpage`. These fields are derived from the original database file, but have been reformatted. See the *Default Fields* section below.

15. 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`.

16. 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.

17. In the `OPTIONS` section of the file are the formatting options. None of these definitions are required. The complete list is given in the *Options keywords* section below, together with explanations of each.

18. The SPECIAL-TEMPLATES section is where users can define their own fields that get generated for every database entry evaluated. For example, the variable definition:

```
group = [<organization>|<institution>|<corporation>|]

in the ``SPECIAL-TEMPLATES`` section will create a ``group`` field that can be used as the varia

Another example usage would be the following:::

author = [<>|<author-en>|]

where the ``author`` field is actually redefined to include not only the *existing* author field
```

19. The order in which any definitions are placed within the special templates is important. For example, if a user has `au = <authorlist.format_authorlist()>` and then below that defines `authorlist = <author.to_namelist()>`, then the code will issue an error stating that `authorlist` is not defined when attempting to create the `au` variable. Since the definition for `au` assumes the presence of the `authorlist` variable, the latter definition must be placed above it.

2.2 Default Fields

A complete of the existing default fields is::

```
au
authorlist
citekey
citenum
ed
editorlist
```

Each of these default fields are defined as "special templates". If a user defines a special template with the same name as one of the above, then the default is overwritten with the user's version. The definitions of these six default special templates are::

```
authorlist = <author.to_namelist()>
editorlist = <editor.to_namelist()>
citelabel = <citenum.remove_leading_zeros()>
sortkey = <citenum>
au = <authorlist.format_authorlist()>
ed = <editorlist.format_editorlist()>
```

Note that the ordering of definitions is important. The following summarizes what these definitions are used for.

authorlist creates a list of dictionaries (one dictionary for each author name found within the database entry's `author` field). Each name dictionary has keys "first", "middle", "prefix", "last", and "suffix", where each of these keys is optional except for "last". Thus, a user can access the first and last name of the first author in the database entry using `<authorlist.0.first> <authorlist.0.last>`. To access the middle name(s) of the second author, use `<authorlist.1.middle>`.

editorlist behaves exactly as `authorlist` but derives its list of names from the database entry's `editor` field rather than `author` field.

citelabel is the thing that appears at the front of the formatted reference, and is identical to the citation label used in the manuscript to point to the item in the reference list. In technical journal articles, this is typically just a number, as in the default definition `<citenum.remove_leading_zeros()>`. The number used here for the label indicates the order in which the entry was cited. Since the variable `citenum` is a string that contains leading zeros, so that the entries are properly sorted in order of 001, 002, ..., 009, 010, 011, ... and not in the strict alphabetical

order of 1, 10, 11, ..., 19, 2, 20, 21, ... But for a citation label, the leading zeros are unsightly, and so we use the `.remove_leading_zeros()` operator to remove them from the string before creating the label.

sortkey is the string used to sort the entry within the reference list. For technical journal articles, what is generally wanted is just the citation order, as indicated by the `<citenum>` variable.

au is the string representing the formatted list of author names. In the default definition shown above, the name list is a standard form, and so simply uses the `.format_authorlist()` operator. Generally, this operator creates name lists that have the form "firstauthor" for only one author, "firstauthor and secondauthor" if only two authors, "firstauthor, secondauthor, ..., and lastauthor" if more than two authors but less than the maximum, and "firstauthor, secondauthor, ..., minauthor, et al." if more than the maximum allowed number of authors. Which author in the list is "minauthor" is defined using the `minauthors` option keyword. The maximum number of allowed authors is set by the `maxauthors` option keyword.

ed follows the same basic structure as `au`, but uses the `maxeditors` and `mineditors` keywords.

2.2.1 Operators

One can use the "dot" operator inside a variable name, as in `<authorname.0.last.initial()>"` to perform any one of four functions: a explicit numerical index (the ``0 shown here), an implicit numerical index (using `.n` or `.N`, for which see section *Examples for namelist formatting* below for details), a dictionary lookup (the `last` used here), or the application of an operator (in this case, the `.initial()` operator which is used to reduce a name to its initial). A numerical index must apply to a list-type of variable, and a key index must apply to a dict-type of variable (i.e. a dictionary).

The complete list of operators available is::

```
.compress()
.format_authorlist()
.format_editorlist()
.frenchinitial()
.if_singular(var1,var2,var3)
.initial()
.monthabbrev()
.monthname()
.ordinal()
.remove_leading_zeros()
.sentence_case()
.tie()
.to_namelist()
```

The function of each operator is summarized below.

.compress() removes any whitespace found within the string. This is useful for generating namelists where the format requires "tight" spacing. An example would be "RMA Azzam", where the three initials are grouped together without spacing. And example template for generating this type of name would be:

```
&lt;authorlist.0.first&gt;&lt;authorlist.0.middle.initial().compress()&gt; &lt;authorlist.0.last&gt;
```

Without the `.compress()` operator, the name would come out as "RM A Azzam", where the two middle name initials "M" amnd "A" are spaced apart from one another by default.

.format_authorlist() operates on a list of dictionaries type of variable (a namelist), and uses the keyword-based default formatting scheme to create a formatted string of names. The complete list keywords that it work with is: `etal_message`, `maxauthors`, `minauthors`, `namelist_format`, `period_after_initial`, `terse_inits`, `use_firstname_initials`, `use_name_ties`. The default formatter, while fast, is not very flexible, so that users looking for more customizability will want to make use of Bibulous' implicit-index and implicit-loop based definitions. See the *Example definitions for namelist formatting* section below.

.format_editorlist() operates on a list of dictionaries type of variable (a namelist), and uses the keyword-based default formatting scheme to create a formatted string of names. The complete list keywords that it work with is: `etal_message`, `maxeditors`, `mineditors`, `namelist_format`, `period_after_initial`, `terse_inits`, `use_firstname_initials`, `use_name_ties`. (The difference with the `.format_authorlist()` operator is that it uses `maxeditors` and `mineditors` rather than `maxauthors` and `minauthors`) The default formatter, while fast, is not very flexible, so that users looking for more customizability will want to make use of Bibulous' implicit-index and implicit-loop based definitions. See the *Example definitions for namelist formatting* section below.

.frenchinitial() is an alternative form of the `.initial()` operator that has slightly different behavior. If a name begins with one of the digraphs "Ch", "Gn", "Ll", "Ph", "Ss", or "Th", then the initial will truncate the name after the digraph instead of after the first letter.

.if_singular(var1,var2,var3) is an operator which inserts `var2` if `var1` has only one element, but `var3` if `var1` has more than one element. Here `var1` is assumed to be a list-type of variable, and `var2` and `var3` are assumed to be either fields present within the database entry or variables defined in the `SPECIAL-TEMPLATES` section of the file.

.initial() will truncate the string to its first letter. Note that if a name begins with a LaTeX markup character, such as `{\'E}`, then the operator will convert the input string to its best attempt at a Unicode-equivalent (without character markup) prior to performing the truncation. Thus, applying the `.initial()` operator to the name `{\v{Z}}ukauskas` will produce the initialized form "Ž".

.monthabbrev() assumes that the input field is a number from 1 to 12, and converts the numerical input into the abbreviated month according to the user's current locale. If the system cannot determine the user's locale, the operator will default to using the American English locale, which replaces the numerical field operated on with one of "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", or "Dec" according to the field's value. Thus, if the bibliography database entry has a field `month = 11`, and the template has the form `<month.monthabbrev()>`, then the template will be replaced with "Nov" for the default locale. For users with locale "Japan", this same operator will return "11".

.monthname() behaves much like `.monthabbrev()` but rather than using an abbreviated form for the month's name, it uses the full form. Thus if the bibliography database entry has a field `month = 3`, and the template has the form `<month.monthname()>`, then the template variable will be replaced with "March" for the default locale. For users with locale "Norway", this same operator will return "Mars".

.ordinal() creates an "ordinal" from a numerical field. Thus, if the field operated on is "1", "2", "3", or "4", then the operator will replace the template with "1st", "2nd", "3rd" or "4th". Any number above 4 simply has "th" appended to the end of it. Currently Bibulous does not support non-English locales for this function. (Anyone having suggestions of how this may be implemented without too much fuss should contact us!)

.remove_leading_zeros() deletes any zeros from the front of the field operated on. Thus "003" will be returned as "3".

.sentence_case() reduces the lower case any characters in the field, except for the initial letter and any letters protected within a pair of curly braces. For example, if the database entry has `title = {Understanding Bohmian mechanics}` and the template has the form `<title.sentence_case()>`, then the template variable will be replaced with "Understanding bohmian mechanics". However, if the entry has `title = {Understanding {B}ohmian mechanics}`, the result will be "Understanding {B}ohmian mechanics".

.tie() replaces any spaces with an unbreakable space. Thus, "R. M. A." becomes "R.~M.~A.". An example use of this operator would be the following template::

```
authorname = [<authorlist.n.first.initial()>.~][<authorlist.n.middle.initial().tie()>. ]...
             [<authorlist.n.prefix>~]<authorlist.n.last>[, <authorlist.n.suffix>]
```

.to_namelist() parses the field (assumed to be a BibTeX-format "and"-delimited list of names) into a Bibulous-format namelist (i.e. a list of dictionaries).

2.3 Options keywords

A complete list of existing options keywords, together with their default definitions, is::

```
allow_scripts = False
backrefs = False
backrefstyle = none
bibitemsep = None
case_sensitive_field_names = False
edmsg1 = , ed.
edmsg2 = , eds
etal_message = , \\textit{et al.}
maxauthors = 9
maxeditors = 5
minauthors = 9
mineditors = 5
namelist_format = first_name_first
period_after_initial = True
procspie_as_journal = False
show_urls = False
sort_case = True
terse_inits = False
undefstr = ???
use_abbrevs = True
use_citeextract = True
use_firstname_initials = True
use_name_ties = False
```

Each of the keywords is summarized below.

allow_scripts [default value: False] tells Bibulous whether to allow the evaluation of Python code in the VARIABLES and DEFINITIONS sections of `.bst` files. It is important for users to realize that evaluating external code in this way is a security risk, and so they should not set `allow_scripts = True` when inserting code that they do not trust. However, as an additional security precaution, Bibulous prevents most security-sensitive operations from being used within its Python API.

backrefs [default value: False] THIS KEYWORD IS NOT YET IMPLEMENTED

backrefstyle [default value: none] THIS KEYWORD IS NOT YET IMPLEMENTED

bibitemsep [default value: None] provides users a means to change the amount of vertical separation that LaTeX sets between entries in the reference list. For example, users wanting a more compact list can define `bibitemsep = 0pt`.

case_sensitive_field_names [default value: False] tells Bibulous whether to consider, for example, a field named "Author" as being distinct from "author".

edmsg1 [default value: , ed.] provides a string to use after a list of editor names, for the case when only one editor is present.

edmsg2 [default value: , eds] provides a string to use after a list of editor names, for the case when multiple editors are present.

etal_message [default value: , \textit{et al.}] provides a string to use after a truncated namelist (for example, when the number of authors exceeds the value given by the `maxauthors` keyword).

maxauthors [default value: 9] provides the maximum number of allowed names in the formatted list of authors. If the number of names is more than this, then the list of names is truncated to `minauthors` and the `etal_message` is appended to the result. (This keyword is only used within the `.format_namelist()` operator.)

maxeditors [default value: 5] provides the maximum number of allowed names in the formatted list of editors. If the number of names is more than this, then the list of names is truncated to `mineditors` and the `etal_message` is appended to the result. (This keyword is only used within the `.format_namelist()` operator.)

minauthors [default value: 9] provides the minimum number of author names to use when truncating an overlength author name list. (This keyword is only used within the `.format_namelist()` operator.)

mineditors [default value: 5] provides the minimum number of editor names to use when truncating an overlength author name list. (This keyword is only used within the `.format_namelist()` operator.)

namelist_format [default value: first_name_first, allowed values: {first_name_first, last_name_first}] defines how the formatted list of names should appear. If `namelist_format = first_name_first` then the individual names will appear in the order "firstname middle prefix last, suffix". If `namelist_format = last_name_first` then the individual names will appear in the order "prefix last, firstname middle, suffix". (This keyword is only used within the `.format_namelist()` operator.)

period_after_initial [default value: True] tells the `.format_namelist()` operator whether to place a period after each initial of an individual's name. Thus, if `period_after_initial = True`, a name will appear as "R. M. A. Azzam", but if `False` will appear as "R M A Azzam". (This keyword is only used within the `.format_namelist()` operator.)

procspie_as_journal [default value: False] The "Proceedings of SPIE" are treated as special by the journals of the Optical Society of America. That is, they format these proceedings (and only these) in the same way that they do journal articles. Thus, a special keyword is required to allow this behavior.

show_urls [default value: False] informs Bibulous whether or not to use the `hyperref` package for placing hyperlinks into the formatted reference.

sort_case [default value: True] informs Bibulous whether or not to use case-sensitive sorting of reference keys.

terse_inits [default value: False] tells the `.format_namelist()` operator whether to compress together the initials of an individual's name. Thus, if `terse_inits = True`, a name will appear as "RMA Azzam", but if `False` will appear as "R. M. A. Azzam". (This keyword is only used within the `.format_namelist()` operator.)

undefstr [default value: ???] informs Bibulous what kind of warning message to print when a required field is missing in the database entry.

use_abbrevs [default value: True] tells Bibulous whether or not to use the abbreviations defined in the bibliography database. (Used for debugging.)

use_citeextract [default value: True] tells Bibulous whether to perform "citation extraction", which creates a small database of only the cited items from among the complete database provided in the `.aux` file.

use_firstname_initials [default value: True] Whether or not to initialize the first names of authors in the formatted authors list. (This keyword is only used within the `.format_namelist()` operator.)

use_name_ties [default value: False] Whether or not to replace spaces with unbreakable spaces (i.e. "R. M. A. Azzam" or "R.~M.~A. Azzam") inside names in the name list. (This keyword is only used within the `.format_namelist()` operator.)

2.4 Examples for namelist formatting

The following code provides an example usage of implicit indexing within an implicit loop structure::

```
authorlist = <author.to_namelist()>
editorlist = <editor.to_namelist()>
authorname.n = [<authorlist.n.first.initial()>. ][<authorlist.n.middle.initial()>. ]...
               [<authorlist.n.prefix> ]<authorlist.n.last>[, <authorlist.n.suffix>]
au = <authorname.0>, ...,{ and }<authorname.9>
```

```
editorname.n = [<editorlist.n.first.initial()>. ][<editorlist.n.middle.initial()>. ]...
               [<editorlist.n.prefix> ]<editorlist.n.last>[, <editorlist.n.suffix>]
ed = <editorname.0>, ...,{ and }<editorname.2>
```

Here the `authorlist` and `editorlist` definitions create namelist variables from the `author` and `editor` fields in the entry (if they exist). Next, the implicitly-indexed `authorname.n` cannot operate except within an implicit loop, and so we should describe that first. It is easier to describe the functionality of the `ed` template than the `au` one, as it has a smaller number of allowed names. The `ed` template has the definition:

```
<editorname.0>, ...,{ and }<editorname.2>
```

which simplifies to `<editorname.0>` when there is only one editor in the database entry, and:

```
<editorname.0> and <editorname.1>
```

when there are only two. Here the separator '' and '' comes from the `{ and }` placed at the right hand side of the implicit loop. For three editors, the implicit loop expands the template to:

```
<editorname.0>, <editorname.1>, and <editorname.2>
```

where this time the comma alone is used as the first delimiter, as it is outside the enclosed braces. For the final element, both the comma and the `{ and }` at the right hand side of the implicit loop are used as the final delimiter. Since the template does not specify the format for more than three editor names, the code builds an *et al.* construction when there more than this number of names, so that the result becomes:

```
<editorname.0>, <editorname.1>, <editorname.2>, \textit{et al.}
```

where the form of the string `\textit{, et. al}` is specified by the `etal_message` keyword option.

Thus, the implicit loop has filled out a unique template based on the number of editors it finds within the database entry. The next step is to use the implicitly-indexed `editorname` to complete building out the template. The latter template is defined as:

```
editorname.n = [<editorlist.n.first.initial()>. ][<editorlist.n.middle.initial()>. ]...
               [<editorlist.n.prefix> ]<editorlist.n.last>[, <editorlist.n.suffix>]
```

so that a template variable of the form "<editorname.0>" is replaced with:

```
[<editorlist.0.first.initial()>. ][<editorlist.0.middle.initial()>. ]...
[<editorlist.0.prefix> ]<editorlist.0.last>[, <editorlist.0.suffix>]
```

That is, the implicit index `.n` is everywhere replaced with the explicit index `0`. For the case of a database entry containing two editor names, the final template will thus have the form:

```
[<editorlist.0.first.initial()>. ][<editorlist.0.middle.initial()>. ]...
[<editorlist.0.prefix> ]<editorlist.0.last>[, <editorlist.0.suffix>] and ...
[<editorlist.1.first.initial()>. ][<editorlist.1.middle.initial()>. ]...
[<editorlist.1.prefix> ]<editorlist.1.last>[, <editorlist.1.suffix>]
```

With this template now complete, the code begins to evaluate the entry and replace the individual variables with their corresponding database fields.

2.5 Python API

Bibulous also provides to users an extensible Python interface allowing users to directly manipulate Bibulous' internal data structures. These use the `VARIABLES` and `DEFINITIONS` sections of the file, as shown below. For the `VARIABLES` section, a variable name is defined (the first example below defines the variable `year_bce`, while the second example below defines `pagerange`). On the right hand side of the definition, however, is a Python function

call. This is different from the other sections of the BST file, which use template syntax. Any variable defined in this way within the VARIABLES section can then be accessed as a template variable (i.e. <year_bce>) within the TEMPLATES section of the file. Two example uses are shown below.

To allow Bibulous to read the VARIABLES and DEFINITIONS sections of the file, users must set the option keyword allow_scripts to True.

First example: a custom yearstyle. For a bibliography containing works from authors dating from before year 0, a common approach is to append "BC" to the year number, and for positive-numbered years, appending "AD". More recently, the convention has been to append "BCE" and "CE" rather than "BC" and "AD". The example defines an option keyword yearstyle that allows users to switch between one style (BC/AD) and the other (BCE/CE). This keyword is accessed by placing options as an argument to the format_yearstyle() function defining the variable year_bce. Inside the function, it can then check the options dictionary for the yearstyle keyword and determine which convention to use.

The format_yearstyle() function itself is straightforward. It first checks whether the entry has a year field. If not, then it returns None, indicating that the function's result is undefined. If it finds a year field, then it checks to see whether it corresponds to an integer. If not, then it returns the field as-is. (Perhaps a user defines his year fields as 45 BCE with the BCE already written out inside the field?) If it finds an integer value, then it determines which style to use (BC/AD or BCE/CE). If the year number is negative then it appends "BC" or "BCE to the end. If the year number is positive then it appends "AD" or "CE to the end, depending on the convention chosen.

Example:

```
OPTIONS:
allow_scripts = True
yearstyle = BCE/CE

VARIABLES:
year_bce = format_yearstyle(entry, options)

DEFINITIONS:
## NOTE! Only Unix-style line endings are allowed here.
def format_yearstyle(entry, options):
    '''
    Append "BC or "AD" to "year", depending on whether the year is positive or negative.
    If the option "yearstyle" is set to "BCE/CE", then use "BCE" and "CE" instead of "BC"
    and "AD".
    '''

    if ('year' not in entry):
        return(options['undefstr'])

    ## First check that the year string is an integer. If not an integer, then just return
    ## the field itself.
    if not str_is_integer(entry['year']):
        return(entry['year'])

    yearnum = int(entry['year'])

    if (yearnum < 0):
        if (options['yearstyle'] == 'BCE/CE'):
            suffix = 'BCE'
        else:
            suffix = 'BC'
        ## The "[1:]" here removes the minus sign.
        result = str(yearnum)[1:] + ' ' + suffix
    elif (yearnum == 0):
        result = str(yearnum)
```

```
    else:
        if (options['yearstyle'] == 'BCE/CE'):
            suffix = 'CE'
        else:
            suffix = 'AD'
        result = str(yearnum) + ' ' + suffix

    return(result)
```

Second example: a custom pagestyle. For a bibliography containing works from magazines, it is not uncommon to find articles with large gaps in page numbers. Here is an example bibliography database entry:

```
@article{stewart,
title = {Interview with Walter Stewart},
author = {Doug Stewart},
journal = {Omni},
year = {1989},
volume = {11},
number = {5},
pages = {64--66, 87--92, 94}
}
```

where we can see that the article was broken into three sections in order to fit the editors' formatting requirements. Many bibliography styles require a starting and ending page, but these are misleading when the article is broken across pages in this way. Thus, a user may want to have the option that if a comma is found within the `pages` field of an entry then it should be displayed as-is. If no comma is found, then it simply returns the standard startpage–endpage pair.

To make this work, first the option `allow_scripts` must be set to true. Next, a new `pagerange` variable is defined, so that it can be accessed in the `TEMPLATES` section of the file as `<pagerange>`. The variable is defined as the return value of the function `format_pagerange()` given in the `DEFINITIONS` section. The defined function first checks to see if there is a `pages` field defined in the entry. If not, then it returns None, so that the `pagerange` variable will also be undefined. If it finds the `pages` field, it looks to see if there is a comma present. If so, it returns the field as-is. If not, it looks for the `endpage` variable (generated by default by Bibulous from the `pages` field). If present, then the function returns a startpage–endpage pair. If `endpage` is not present, then it returns only the `startpage` variable.

Example:

```
OPTIONS:
allow_scripts = True

VARIABLES:
pagerange = format_pagerange(entry, options)

DEFINITIONS:
def format_pagerange(entry, options):
    '''
    If the "pages" field is comma-delimited, then return the pages field as-is. Otherwise
    return the standard startpage--endpage range.
    '''

    if not ('pages' in entry):
        return(None)
    elif (',' in entry['pages']):
        return(entry['pages'])
    elif ('endpage' in entry):
        return(entry['startpage']--entry['endpage'])
```

```
    else:
        return(entry['startpage'])
```

CHAPTER

THREE

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

3.1 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.)

3.2 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

FOUR

DEVELOPER GUIDE

4.1 Guidelines and notes for Python coding style

1. Note that one 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 120 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, but Bibulous is currently organized in a single large file. This is partly because there is no large block of code that fits by itself so that a separate file makes sense. (Parsing of `.bib` files, for example, only requires a couple hundred lines.

4.2 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, together with the entire set of citations.
2. Read in the Bibulous style template file as a dictionary (`bstdict`).
3. If the `use_citeextract` keyword is set to True, and if an "extracted" database file exists, then compare the citations in the extracted database against those in the `.aux` file. If there are any differences, then re-extract the database. Otherwise, use the extracted database rather than the full one specified in the `.aux` file.
4. Read in all of the bibliography database files into one long dictionary (`bibdata`), replacing any abbreviations with their full form. In an "extracted" database, all entries are parsed, whereas in any other type of database file, only those entries whose keywords are found in the citation list are actually parsed. All other entries have their data saved as unparsed strings. 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. It is only then that the cross-referenced entries have their data parsed into dictionary form.

5. Now that all the information is collected, we can generate the `.bbl` file. Create the list of sortkeys, then go through each corresponding citation key in turn, and find the corresponding entry key in `bibdata`. If there is crossref data, then fill in missing values here. Also create the "special variables" 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 `Bibdata` class thus holds all relevant information needed to operate on a bibliography and generate the output LaTeX-formatted `.bbl` file.

4.3 Parsing BIB files

4.3.1 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 further parsing. To gather the entry data string, we first look for a 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 for the BibTeX format, since it allows nested delimiters, is not directly compatible with regular expressions.

4.3.2 parse_bibentry()

Because `parse_bibfile()` has already split the data by individual entry, `parse_bibentry()` only needs to worry about parsing a single entry, and there are five 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 `acronym`, then get the entrykey and copy it into the `name` field. The remainder of the string is a single field value (the full form of the acronym); copy that into the `description` field.
4. 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.
5. 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), it locates each individual field and separates out the string corresponding to the key-value pair for each field. It does not actually *parse* the individual fields. For that, it loops over each field with a call to `parse_bibfield()` to extract the field key-value pairs.

4.3.3 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 unnested `"`. 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 for 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 continue on.

4.4 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 costs 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.

4.5 Parsing BST files

Parsing a `.bst` file basically involves looking for one of several syntactical structures.

1. First, any `#` present in a line indicates a comment. All text following the `#` are ignored.
2. Any line containing all capital letters and ending in `:` indicates a section header. The sections recognized are: `TEMPLATES`, `SPECIAL-TEMPLATES`, `OPTIONS`, `VARIABLES`, and `DEFINITIONS`. The first three sections (`TEMPLATES`, `SPECIAL-TEMPLATES`, and `OPTIONS`) use template syntax, while the last two (`VARIABLES` and `DEFINITIONS`) use Python syntax.
3. In the TEMPLATES, SPECIAL-TEMPLATES, or OPTIONS sections of the file, any line ending in an ellipsis (`...`) means that the following line is a continuation. Thus, the following line is appended to the current one.
4. For each `var = definition` pair found in the `VARIABLES` section of the file, the code creates a new entry in the `user_variables` dictionary, with value equal to the given definition.
5. For each `entrytype = template` pair found in the `TEMPLATES` section of the file, the code creates a corresponding entry in `bstdict`, with the key given by the `entrytype` and value given by the `template`. The code next examines the template definition to see if it contains a nested options block. If so, it adds it to the list of nested templates.

6. For each `keyword = value` pair found in the `OPTIONS` section of the file, the code creates a new entry in the `options` dictionary, with the dictionary key being the keyword itself, and the value copied from the right hand side of the option definition.
7. For each `var = definition` pair found in the `SPECIAL-TEMPLATES` section of the file, the code has to do a little more work than elsewhere. First it creates a new entry in the `specials` dictionary, with the dictionary key given by the `var`, and the value given by the `definition`. It then appens the key to the `specials_list`. (Since a dictionary is not ordered, we need an order-preserving means of iterating through the list of specials to make sure that one can always be defined before another that depends on it.) Next it examines the template definition to see if it contains a nested options block. If so, it adds it to the list of nested templates. It also looks to see if there is an ellipsis representing an implicit loop. If so, it adds the template key to the list of "looped templates". Finally it looks to see if the template's key represents an inmplicitly-indexed variable. If so, it adds the key to the list of implicitly indexed variables.

Once the initial parsing is done, there are several steps in which it analyzes the results:

1. Iterating through each of the regular templates, the code looks to see if any of the templates are defined as copies of other templates, as, for example, `inbook = incollection`. If it finds this kind of definition, then it copies the template from the one (`incollection` here) to the other (`inbook` here).
2. The code looks at the functions defined in the `DEFINITIONS` section of the file. If the `allow_scripts` keyword is set to True, then it goes ahead and evaluates these function definitions so that they will be available during the process of formatting bibliography entries.
3. Finally, the code passes each template definition through the `validate_templatestr()` function to validate that the template has proper syntax.

4.6 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. Then, for each citation key we found in the `.aux` file, we

1. Insert any cross-reference data from any other database entries into the current one.
2. Define all of the "special variables", including the `sortkey` and `citelabel`, as fields within the current entry.

Now that we have all of the sortkeys, we generate the `citation_list` — the thing we iterate through one by one to format the references in order. At each iteration, we call `format_bibitem()`, which does the following:

1. Write the line `\bibitem[citelabel]{citekey}` into the `.bbl` file.
2. Import the template corresponding to the current entry's `entrytype`.
3. If there are any user-deefined variables (from the `VARIABLES` section of the file), then evaluate those variables now, so that they can be used inside the template.
4. For each option block in the template, go through and determine how to "simplify" the block. This amounts to locating the first cell in each block that has a defined value, and then replacing the `[...]` square-bracket-delimited block with its contents. At this point the template variables are still there; only the square brackets have been dropped.
5. Now that the optional pieces are all gone, go through each template variable and replace it with the corresponding field from the database entry.
6. If there are any nested `\textit{...\textit{...}...}` operators in the result, replace odd-level operators with `\textup{...}` in order to get the right behavior of flipping between italics and regular font.

7. If there are any nested `\textbf{...\textbf{...}...}` operators in the result, replace odd-level operators with `\textup{...}` in order to get the right behavior of flipping between bold and regular weight.
8. If there are any nested quotation marks in the result, then re-order them according to the American standard. This means having double-quotation-marks at the outermost level, single-quotation-marks inside that, then double inside that, single inside that, and so on. This is messy and difficult code, and so users should always be recommended to use the `\enquote{...}` LaTeX operator instead of manually-implemented quotation marks.

4.7 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()`.

4.7.1 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.

4.7.2 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.

4.8 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.

4.9 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:

   ```
   @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:

   ```
   @initialize1{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. `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 `entrytype` is the key given in the new entry of the BIB file you just put in (e.g. `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.

4.10 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

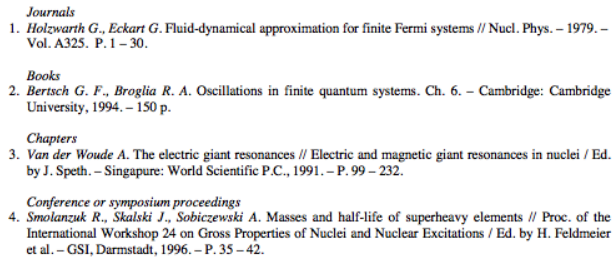
FIVE

EXAMPLES

5.1 Example 1

The following example is taken from a question posted at `http://tex.stackexchange.com/questions/147675/bibtex-` The OP asks:

> Where can I find bibtex style, which looks approximately like this? Somehow all styles (even `science.bst`), which I find use `format.vol.num.pages` function, which put pages right after volume number. But the order I used to see and use is `volNum-year-pages`. I can simply move the line `format.date "year" output.check` into `format.vol.num.pages`, but 1. I am not sure whether this is correct (looks strange for me), 2. I don't know how to add the `---` sign in front.



Journals

1. *Holzwarth G., Eckart G.* Fluid-dynamical approximation for finite Fermi systems // Nucl. Phys. – 1979. – Vol. A325. P. 1 – 30.

Books

2. *Bertsch G. F., Broglia R. A.* Oscillations in finite quantum systems. Ch. 6. – Cambridge: Cambridge University, 1994. – 150 p.

Chapters

3. *Van der Woude A.* The electric giant resonances // Electric and magnetic giant resonances in nuclei / Ed. by J. Speth. – Singapure: World Scientific P.C., 1991. – P. 99 – 232.

Conference or symposium proceedings

4. *Smolanzuk R., Skalski J., Sobiczewski A.* Masses and half-life of superheavy elements // Proc. of the International Workshop 24 on Gross Properties of Nuclei and Nuclear Excitations / Ed. by H. Feldmeier et al. – GSI, Darmstadt, 1996. – P. 35 – 42.

With Bibulous, we can easily provide templates that provide the formatting that the OP asks for::

```
TEMPLATES:
article = \textit{<au>} <title> // <journal> -- <year>. -- Vol.~<volume>. P.~[<startpage> -- <endpage
book = \textit{<au>} <title>. [Ch.~<chapter>. ]-- <address>: <publisher>, <year>.[ -- <startpage>~p.]
inbook = \textit{<au>} <title> // <booktitle>[ / Ed. by <ed.if_singular(editorlist, nothing, etal_mes
         <publisher>, <year>.[ -- P.~<startpage> -- <endpage>| -- P.~<startpage>| -- P.~<eid>|].
inproceedings = \textit{<au>} <title> // <booktitle>[ / Ed. by <ed.if_singular(editorlist, nothing, e
         <publisher>, <address>, <year>. -- P.~[<startpage> -- <endpage>|<startpage>|<eid>|].

SPECIAL-TEMPLATES:
authorlist = <author.to_namelist()>
editorlist = <editor.to_namelist()>
authorname.n = [<authorlist.n.prefix> ]<authorlist.n.last>[ <authorlist.n.first.initial()>.][ <autho
au = <authorname.0>, ..., <authorname.9>
ed = [<editorlist.0.first.initial()>. ][ <editorlist.0.middle.initial()>. ][<editorlist.0.prefix> ]<e

OPTIONS:
nothing = {}
```

so that if we use this template together with the following database file:

```
@article{Holzwarth,
  author = {G. Holzwarth and G. Eckart},
  title = {Fluid-dynamical approximation for finite Fermi systems},
  journal = {Nucl. Phys.},
  year = 1979,
  volume = {A325},
  pages = {1-30}
}
@book{Bertsch,
  author = {G. F. Bertsch and R. A. Broglia},
  title = {Oscillations in finite quantum systems},
  chapter = 6,
  address = {Cambridge},
  publisher = {Cambridge University},
  year = 1994,
  pages = 150
}
@inbook{Woude,
  author = {A. Van der Woude},
  title = {The electric giant resonances},
  booktitle = {Electric and magnetic giant resonances in nuclei},
  editor = {J. Speth},
  address = {Singapore},
  publisher = {World Scientific P.C.},
  year = 1991,
  pages = {99-232}
}
@inproceedings{Smolanzuk,
  author = {R. Smolanzuk and J. Skalski and A. Sobiczewski},
  title = {Masses and half-life of superheavy elements},
  booktitle = {Proc.\ of the International Workshop 24 on Gross Properties of Nuclei and Nuclear Exci
  editor = {H. Feldmeier and others},
  address = {Darmstadt},
  publisher = {GSI},
  year = 1996,
  pages = {35-42}
}
```

then we get the formatted result shown below

References

1. *Holzwarth G., Eckart G.* Fluid-dynamical approximation for finite Fermi systems // Nucl. Phys. – 1979. – Vol. A325. P. 1 – 30.
2. *Bertsch G. F., Broglia R. A.* Oscillations in finite quantum systems. Ch. 6. – Cambridge: Cambridge University, 1994. – 150 p.
3. *Van der Woude A.* The electric giant resonances // Electric and magnetic giant resonances in nuclei / Ed. by J. Speth. – Singapore: World Scientific P.C., 1991. – P. 99 – 232.
4. *Smolanzuk R., Skalski J., Sobiczewski A.* Masses and half-life of superheavy elements // Proc. of the International Workshop 24 on Gross Properties of Nuclei and Nuclear Excitations / Ed. by H. Feldmeier, *et al.*. – GSI, Darmstadt, 1996. – P. 35 – 42.

5.2 Example 2

The next example is taken from the bibliography style found in: Dimitri Mihalas and James Binney, _Galactic Astronomy: Structure and **Kinematics**_, 2nd ed. (W. H. Freeman, New York, 1981). A snapshot from the the book's bibliography looks like

(B12) Blaauw, A. and Schmidt, M. (eds.). 1965. *Galactic Structure*. (Chicago: University of Chicago Press).

(B13) Bok, B. J. 1977. *Pub. Astron. Soc. Pacific.* **89**:597.

(B14) Bosma, A. 1978. Ph.D. Thesis, University of Groningen, Netherlands.

(B15) Burke, B. F. 1957. *Astron. J.* **62**:90.

(B16) Burton, W. B. 1970. *Astron. and Astrophys.* **10**:76.

(B17) Burton, W. B. 1972. *Astron. and Astrophys.* **19**:51.

(B18) Burton, W. B. 1976. *Ann. Rev. Astron. and Astrophys.* **14**:275.

(B19) Burton, W. B. and Gordon, M. A. 1978. *Astron. and Astrophys.* **63**:7.

(C1) Chiu, H.-Y. and Muriel, A. (eds.). 1970. *Galactic Astronomy*. (New York Gordon and Breach).

(C2) Cohen, R. J. and Davies, R. D. 1976. *Mon. Not. Roy. Astron. Soc.* **175**:1.

(D1) Dickman, R. L. 1978. *Astrophys. J. Supp.* **37**:407.

(E1) Emerson, D. T. 1978. *Astron. and Astrophys.* **63**:L29.

(F1) Fichtel, C. E. and Stecker, F. W. (eds.). 1977. *The Structure and Content of the Galaxy and Galactic Gamma Rays* (Washington, D.C.: NASA).

(F2) Fletcher, E. 1963. *Astron. J.* **68**:407.

To produce this style, we can define the following templates::

```
TEMPLATES:
article = <au> <year>. \textit{<journal>} \textbf{<volume>}:<pages>.
book = [<au>|<ed.if_singular(editorlist, eds_msg1, eds_msg2)>|]. <year>. \textit{<title>}. (<address>
proceedings = book
phdthesis = <au> <year>. Ph.D. Thesis, <university>.

SPECIAL-TEMPLATES:
authorlist = <author.to_namelist()>
editorlist = <editor.to_namelist()>
authorname.n = [<authorlist.n.prefix> ]<authorlist.n.last>[, <authorlist.n.first.initial()>.][ <autho
au = <authorname.0>, ...,{ and }<authorname.9>
editorname.n = [<editorlist.n.prefix> ]<editorlist.n.last>[, <editorlist.n.first.initial()>.][ <edito
ed = <editorname.0>, ...,{ and }<editorname.5>
authorlabel = [<authorlist.0.prefix>|<authorlist.0.last>]
editorlabel = [<editorlist.0.prefix>|<editorlist.0.last>]
sortkey = [<authorlabel><year>|<editorlabel><year>]
citelabel = (<sortkey.initial()>)

OPTIONS:
eds_msg1 = { }(ed.)
eds_msg2 = { }(eds.)
use_citeextract = False
```

so that with the following database file:

```
@book{Blaauw1965,
  editor = {A. Blaauw and M. Schmidt},
  title = {Galactic Structure},
  address = {Chicago},
  publisher = {University of Chicago Press},
  year = 1965
}
@article{Bok1977,
  author = {B. J. Bok},
  journal = {Publ. Astron. Soc. Pacific},
  year = 1977,
  volume = 89,
  pages = 597
}
@phdthesis{Bosma1978,
```

```
  author = {A. Bosma},
  university = {University of Groningen, Netherlands},
  year = 1978
}
@article{Burke1957,
  author = {B. F. Burke},
  journal = {Astron. J.},
  year = 1957,
  volume = 62,
  pages = 90
}
@article{Burton1970,
  author = {W. B. Burton},
  journal = {Astron. and Astrophys.},
  year = 1970,
  volume = 10,
  pages = 76
}
@article{Burton1972,
  author = {W. B. Burton},
  journal = {Astron. and Astrophys.},
  year = 1972,
  volume = 19,
  pages = 51
}
@article{Burton1976,
  author = {W. B. Burton},
  journal = {Ann. Rev. Astron. and Astrophys.},
  year = 1976,
  volume = 14,
  pages = 275
}
@article{Burton1978,
  author = {W. B. Burton and M. A. Gordon},
  journal = {Astron. and Astrophys.},
  year = 1978,
  volume = 63,
  pages = 7
}
@book{Chiu1970,
  editor = {H.-Y. Chiu and A. Muriel},
  title = {Galactic Astronomy},
  year = 1970,
  address = {New York},
  publisher = {Gordon and Breach}
}
@article{Cohen1976,
  author = {R. J. Cohen and R. D. Davies},
  journal = {Mon. Not. Roy. Astron. Soc.},
  year = 1976,
  volume = 175,
  pages = 1
}
@article{Dickman1978,
  author = {R. L. Dickman},
  journal = {Astrophys. J. Supp.},
  year = 1978,
  volume = 37,
```

```
  pages = 407
}
@article{Emerson1978,
  author = {D. T. Emerson},
  journal = {Astron. and Astrophys.},
  year = 1978,
  volume = 63,
  pages = {L29},
}
@inproceedings{Fichtel1977,
  editor = {C. E. Fichtel and F. W. Stecker},
  booktitle = {The Structure and Content of the Galaxy and Galactic Gamma Rays},
  address = {Washington, D.C.},
  publisher = {NASA},
  year = 1977
}
```

we get the following formatted result

CHAPTER

SIX

BIBULOUS 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, increased options for author name formatting, and more. And one can use the same basic structures and LaTeX commands to generating each of: a bibliography, a glossary, an annotated bibliography, a list of acronyms and symbols, and more, 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 Installation

Installing using pip::

```
pip install bibulous
```

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.

6.2 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 = <au>, \enquote{<title>,} \textit{<journal>} \textbf{<volume>}: ...
             [<startpage>--<endpage>|<startpage>|<eid>|] (<year>).[ <note>]
book = [<au>|<ed>|], \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. If a user wishes to add a new field, such as translator, then if it has been added to the .bib database file, then all that is needed is to add some text to the template, such as (<translator>, trans.) to insert the field into every bibliography entry that has translator defined for that entrytype.

6.3 Developers

Bibulous is a new project, and so it has until now 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 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:

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.5 Indices and tables

- *genindex*
- *search*