Bibulous — A drop-in BibTeX replacement based on style templates

Nathan Hagen

Abstract

BibTeX has long been an essential tool for TeX and LaTeX users, but the long list of re-implementation efforts attests to unfulfilled needs. The Bibulous project attempts to fulfill many of these needs with a unique approach based on style templates, providing a flexibility that is unmatched by existing tools. We illustrate its capability with examples of custom bibliography styles, multilingual bibliographies, glossaries, and more.

1 Introduction

For beginning LaTeX/BibTeX users, the difficulty of building and customizing bibliography style files can come as a shock. BibTeX's style files are written in an old-style stack-based language that requires significant effort to learn, edit, and write. It is natural, however, to feel that since bibliographies are highly structured one *should* be able to specify them simply and not, as one commonly finds with BibTeX's bibliography styles, in files with over a thousand lines of difficult code. The Bibulous project shows that with style templates one can indeed specify complete bibliography styles in a matter of only a few lines of text, using a format understandable even for novices. Style templates provide a flexibility unmatched by alternative approaches to creating bibliographies, and have important advantages over even the more modern approach taken by Biblatex [1] and Biber [2, 3].

In addition to style templates, Bibulous also implements many of the modern enhancements to BibTeX, such as the ability to work smoothly with languages other than English, better support for non-standard bibliography structures and heterogeneous databases, and increased formatting options, among others. Moreover, one can use the same framework to generate any of a bibliography, glossary, nomenclature, list of symbols, and/or more, by specifying a different style template for each case.

2 Template example

A short example illustrates how templates work. For a simple bibliography consisting of only journal articles and books, a complete style file may consist of just a few lines, as shown in Figure 1. Angle brackets indicate a template variable, which can be (*a*) a variable (*e.g.* `<year>`) that maps to a field (*e.g.* `year = {·}`) in the database entry; (*b*) a "special" variable that is generated by the program, *e.g.* `<au>` is generated by formatting the database entry's `author = {·}` field into a list of names; or (*c*) a variable that is defined by the user within the template's variable definitions. Each variable represents a string to be inserted into the template at that point, so that for a database entry with `year = {1988}`, a template containing `(<year>)` is replaced with `(1988)`. The same procedure follows for each variable found within the template definition.

Any variable that is not wrapped within square brackets is considered a *required* variable: if this field is missing from the bibliography database entry, then a warning message is used in its place, with the default warning message being simply `???`.

The `[·|·]` bracket notation behaves like an `if ... elseif...` statement: before performing any substitutions of variables into the template, the contents of the first block — located inside square brackets `[·]`, or between the open square bracket and the first vertical bar `[·|` — are checked to see whether all variables within it are defined within the entry. If any variables inside the block are undefined (missing from the database entry), then the second block (between the vertical bar and the close bracket `|·]`, is evaluated. And so on. This provides a flexible way of modifying the inputs to the formatted reference depending on which fields exist within a given database entry. Thus, while some users may prefer to define a given field as `institution`, and others may prefer `organization`, a structure such as

```
[<institution>|<organization>]
```

provides a simple means of accommodating both.

Finally, a structure that ends with an empty block (`|]` rather than `]`) means that even though the individual blocks are optional, at least one of the blocks is required to be defined within the database entry. Thus, for example, the required variable `<note>` is equivalent to `[<note>|]`.

To see how Bibulous uses a template to generate `.bbl` file entries, we can walk through the definitions given in Fig. 1. First, from the `.aux` file, Bibulous obtains a list of citation keys that map to database entries. Using the `sortkey` template, it generates a key for each reference, sorts the keys into the desired order, and then walks through each corresponding database entry, using an appropriate template to format each reference in turn. In the case of the example shown, the sorting key is simply the numbering in which the entry was cited (`<citenum>`). When Bibulous locates an entry in the database, it reads the entrytype (this particular style file assumes that only entrytypes `article` and `book` exist in the citation list) to locate which template definition to use. If

short.bst

```
TEMPLATES:
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>]
SPECIAL-TEMPLATES:
sortkey = <citenum>
citelabel = <citenum>
```

Figure 1: A style template file example. An ellipsis (...) at the end of a line indicates a line continuation. All entries in the "Special Templates" section of the file are optional, and are used to replace default settings.

the current database entry is an `article` entrytype, Bibulous will find the list of variables in the `article` template definition and begin replacing them one by one with their corresponding fields. The `<au>` variable represents a formatted string of author names (formatted from the `author` field according to default options, in this case), followed by a comma and a space. If no `author` field is found in the bibliography entry, then it inserts "`???`" there to indicate a missing required field. Next it locates the `title` field and inserts it, with a comma, inside `\enquote{·}` formatting instructions. Next follows an italicized journal name, and a boldface volume number. All of these so far are designated as required fields. Next, if the `pages` field is found in the entry's database, Bibulous will parse the start and end page numbers (assuming that the `pages` field uses dash-delimited page identifiers) and insert them here using an en-dash separator. If the `pages` field contains only one page, then the `endpage` variable will be undefined, and Bibulous will attempt to use the next block, which uses only the `startpage`. If the `pages` field is missing entirely from the database entry, then Bibulous checks to see if the `eid` (electronic identifier) field is defined, and if so uses it instead. However, if the `pages` and `eid` fields are both undefined, then the code inserts "`???`". Finally, the template instructs putting the `year` inside parentheses, and if a `note` field is defined in the entry, then it is added onto the end (following the period). If `note` is not defined, then Bibulous adds nothing.

The `book` template in the example follows a similar procedure, but has different required and optional fields. The list of names at the beginning of the reference, for example, can be either a list of authors or a list of editors (with preference given to authors, if defined in the entry). The `book` template also requires a `publisher` field to be inserted inside the parentheses with the publication year.

The `sortkey` "special template" allows users flexibility in defining how they want their citations sorted. The default is simply to sort by numerical order (`sortkey = <citenum>`), but one can also choose to sort by author name, then title, then year, as in

```
sortkey = <authorlist.0.last>...
          <authorlist.0.first>...
          [<title>|<booktitle>]<year>
```

or, if a user wants the entry to define a special sort for any given entry, then they can use any fields within the entry that they wish. For example, to have an author sorted under a different name than is used in the reference, one can add a `sortname` key to the database entry and redefine the `sortkey` template as

```
sortkey = [<sortname>|<authorlist.0.last>...
          <authorlist.0.first>|][<title>|...
          <booktitle>]<year>
```

And likewise for any other field such as `sorttitle` or `sortyear` for modifying citation sorting.

The final item that Bibulous needs to specify for the reference is its item label — the label that goes at the front of the reference, and is often the same as the `sortkey`. This label is generated using the `citelabel` template. For scientific journal articles the most common choice for this is the citation number (*e.g.* `citelabel = <citenum>`). For other areas of study, the reference labels may, for example, take the form of the first author's last name, then first name, then year, as in the following example:

```
citelabel = \textbf{<authorlist.0.last>, ...
            <authorlist.0.first> (<year>).}
```

This can produce a reference list like the following:

Schmader, Toni (2002). Gender identification moderates stereotype threat effects on women's math performance. *Journal of Experimental Social Psychology*, **38**, 194–201.

Steele, Claude (1997). A threat in the air: How stereotypes shape intellectual identity and performance. *American Psychologist*, **52**, 613–629.

ieee.bst

```
TEMPLATES:
article = <au>. \enquote{[\href{<url>}{<title>}|\href{<doi>}{<title>}|<title>|],} ...
       \textit{<journal>} <volume>(<number>): [<startpage>--<endpage>|...
       <startpage>|<eid>|] (<year>).
inproceedings = <au>, \enquote{[\href{<url>}{<title>}|\href{<doi>}{<title>}|...
       <title>|],} in \textit{<booktitle>}[, <ed.if_singular(editorlist, edmsg1, ...
       edmsg2)>][, in <series>][, vol.~<volume>][, pp.~<startpage>--<endpage>|, ...
       <eid>] (<year>).
SPECIAL-TEMPLATES:
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.3>
OPTIONS:
etal_message = , \textit{et~al.}
edmsg1 = , ed.
edmsg2 = , eds
```

Figure 2: Two example entrytype templates for an IEEE format bibliography. For the hyperlinked title, the template allows use of either a `url` or a `doi` field in the database entry.

Figure 2 shows how Bibulous can be used to develop custom bibliography styles, using an example that incorporates hyperlinks into the document title of each reference. In going from the style used in Fig. 1 to the new `article` style in Fig. 2, we have to convert from using italics for titles to using quotation marks around a hyperlinked text (if a `url` or `doi` field is provided in the entry), from using boldface to standard typeface for the volume, and adding an issue number in parentheses. The `inproceedings` style uses the same format for hyperlinked titles, and adds the ability to format fields such as `booktitle` and `series`, and provides a separate list of editors. For BibTeX, modifying an existing style file to achieve these changes would be a daunting task for a casual user. Biblatex has succeeded in making this much easier, but with templates it can be easier still.

The other major change in going from Fig. 1 to Fig. 2 is in the special template definitions, where we can find several new constructions. The first is a "dot" operator placed after a variable inside the angle brackets. This can be used to apply an index — as in the case of a list or a dictionary type of variable — or an operator, as in `.to_namelist()` or `.initial()`. An index can be *explicit*, as in `<authorlist.0>`, which indicates taking the zeroth element of the list-type variable `authorlist`, or *implicit*, as in `editorname.n`. For an implicit index, the construction `<editorname.0>` used in conjunction with the variable definition for `editorname.n` indicates that all instances of the implicit index `n` in the template should be replaced with the explicit index `0`. Finally, an ellipsis occurring in the *middle* of a string indicates an implicit loop. Thus, the definition

```
<editorname.0>, ...,{ and }<editorname.3>
```

is equivalent to `<editorname.0>` when there is only one editor in the database entry, or equivalent to

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

when there are only two. For three editors, the implicit loop expands the template to

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

and for four,

```
<editorname.0>, <editorname.1>,
<editorname.2>, and <editorname.3>
```

Since the template does not specify the format for more than four editor names, the code builds an *et al.* construction when there more than the specified number of names, so that the result becomes

```
<editorname.0>, <editorname.1>,
<editorname.2>, <editorname.3>,
\textit{et~al.}
```

where the form of the string "`, et al.`" is specified by an `etal_message` keyword option.

The implicit index and implicit loop features are only necessary when one needs to customize the

moviedb.bib

```
@movie{key,
  title = {...},
  director = {...},
  year = {...},
  rating = {...}
}
```

moviedb.tex

```
%% In preamble:
\usepackage{expl3}
\ExplSyntaxOn
\cs_new_eq:NN \Repeat \prg_replicate:nn
\ExplSyntaxOff
\newcommand{\nstars}[1]{\parbox{3em}{%
  \Repeat{#1}{$\star$}}}
```

moviedb.bst

```
TEMPLATES:
movie = \nstars{<rating>} \color{blue}{...
   <title>}\color{black}, <director> (\year>).
SPECIAL-TEMPLATES:
editor = <director>
editorlist = <editor.to_namelist()>
sortkey = <-year><title><editorlist.0.last>
citelabel = None
OPTIONS:
bibitemsep = 0pt
```

1. ⋆⋆⋆⋆⋆ The Inheritance, Per Fly (2003).
2. ⋆⋆⋆⋆ The Celebration, Thomas Vinterberg (1998).
3. ⋆ The Kingdom, Lars von Trier (1994).

Figure 3: An example illustrating the use of Bibulous' style templates for a movie database. The citation sorting order is defined to be by year in descending order (the minus sign within the variable name inverts the direction of the sort), followed by title and then director's last name. Note that the `editor = <director>` special template is necessary in order to map the `director` field to the `editor` field.

namelist formats with more flexibility than that allowed by Bibulous' standard keyword arguments. Thus, for most styles these structures are not necessary, and one can use the default namelist format, modified through keyword arguments.

The `.if_singular(arg1, arg2, arg3)` operator is unique in that it accepts arguments. This allows it to insert `arg2` if `arg1` is singular, or `arg3` if plural. Here, the operator is given the argument `<editorlist>`, which has one entry if the `editor` field in the database entry contains only one name. In this case it returns the argument `edmsg1`, defined in the options list as "`, ed.`". If more than one editor name is found in the database entry, then the operator returns the argument `edmsg2`, defined in the options list as "`, eds`".

3 Unorthodox bibliography styles

Templates allow easy customization of how BibTeX-format database entries are displayed. Fig. 3 shows an example `.bib` database structure, and a corresponding template file, for the display of a movie database. This simple example defines a LaTeX command `\nstars{·}` for printing N stars as a display of the `rating` variable, shows the use of color commands, and works with new database fields. Thus, any LaTeX markup the user wishes to incorporate into a bibliography, or into any database formatter, can be implemented in Bibulous' style templates.

Another important feature of style templates is that they are language agnostic. For example, if we replace

```
TEMPLATES:
article = <au>, \enquote{...
book = [<au>|<ed>...
```

from the style template with

```
学術論文 = <au>, \enquote{...
本 = [<au>|<ed>...
```

(学術論文 and 本 are the Japanese words for "journal article" and "book"), then we can use the Japanese forms for every entry in the bibliography database, so that entries in the `.bib` file would have the form

```
@学術論文{entrykey, ...
@本{entrykey, ...
```

Since this shows only the entry types and not the database fields in a non-English language, this still shows an English-language-centric implementation, but here is a fully Japanese style template:

```
本 = <著者名>, <題名> (<年>).
```

with a corresponding example database file entry

```
@本{漱石1906,
   題名 = {草枕},
   著者名 = {夏目 漱石},
   年 = 1906,}
```

The important feature here is that Bibulous does not need to know anything about the languages used in the database, the template, or the `.tex` file. It only needs to be able to convert the symbols to Unicode so that it can match any symbol from one file to that in another. And since Bibulous' internals are written entirely in UTF-8 compliant code, multilingualism and localization come easily. Bibulous is even flexible enough to allow the use of

```
TEMPLATES:
collection = [<for_japanese_audience><author_ja>, <title_ja>[, <volume>巻] (...
   <publisher_ja> <year>).|<for_english_audience><author_en>, <title_ja> ...
   (<title_romaji>) {\makeopenbracket}<title_en>{\makeclosebracket}...
   [, vol.~<volume>] (<publisher>, <address>, <year>).|<for_english_audience>...
   <author>, <title>[, vol.~<volume>] (<publisher>, <address>, <year>)|]
OPTIONS:
document_language = Japanese  ## whether the document is for a Japanese audience
VARIABLES:
for_japanese_audience = '{}' if (options['document_language'] == 'Japanese') else None
for_english_audience = '{}' if (options['document_language'] == 'English') else None
```

Figure 4: Template file for the multi-language bibliography example.

mathematics markup within keys and labels, even though LaTeX itself currently cannot handle this.

Another example of Bibulous' versatility in dealing with multiple languages is that of an "audience switch". The following example is adapted from a query made online [4]. For a single database file that can be used inside both English-language *and* Japanese-language publications, an English-language format should allow Japanese entries in the bibliography with author names in both *kanji* and romanized characters. Titles should also be given in this dual form, but sometimes with an English translation provided. English-language citations should appear as normal. In a Japanese-language publication, however, the Japanese entries should have the author, title, etc. in Japanese form without roman letters or English. Here is an example database entry:

```
@collection{Tsuruta2006,
  address_ja = {東京,
  address_romaji = {Tōkyō}
  author_ja = {鶴田　匡夫},
  author_en = {Tadao Tsuruta}
  publisher_ja = {新技術コミュニケーションズ},
  publisher_en = {Shin'gijutsu
                  Communications},
  title_ja = {光の鉛筆},
  title_romaji = {Hikari no empitsu},
  title_en = {Pencil of light},
  volume = 7,
  year = 2006}
```

The corresponding style template file is shown in Fig. 4. The template is structured into two large optional blocks, and the selection of one or the other is decided by the `document_language` option variable defined by the user within the file. The formatted result will look something like the following for a Japanese audience

1. 夏目 漱石, 草枕 (春陽堂 1906).
2. 柳田 聖山, 禪學叢書, 10巻 (中文出版社 1974–1977).
3. 鶴田 匡夫, 光の鉛筆, 7巻 (新技術コミュニケーションズ 2006).

and the following for an English-language audience

1. Soseki Natsume, 草枕 (Kusamakura) [Pillow of Grass] (Shun'yōdō Publishing, Tōkyō 1906).
2. Seizan Yanagida, 禪學叢書 (Zengaku sōsho) [Collected Materials for the Study of Zen], vol. 10 (Chinese Book Publ. Co., Kyōtō 1974–1977).
3. Tadao Tsuruta, 光の鉛筆 (Hikari no empitsu) [Pencil of Light], vol. 7 (Shin'gijutsu Communications, Tōkyō 2006).

Thus, the template shown in Fig. 4 is not only flexible enough to handle custom database types and heterogeneous fields, but also different audience languages. Note that the example uses fields in English rather than in Japanese (*e.g.* `author_ja` rather than 著者名) in both the database and the template in order to illustrate the overall structure clearly to readers unfamiliar with Japanese.

4 Implementing a glossary

The utility of style templates extends beyond bibliographies. Glossaries, lists of symbols (*i.e.* nomenclatures), and lists of acronyms are readily generated with Bibulous. Figure 5 shows an example `.tex` file, together with corresponding `.bib` and `.bst` files, and output `.bbl` file. Together, these create the formatted glossary etc. shown in Fig. 6.

In the example `.bib` file, readers may note the separation of the `symbol` entrytypes into independent `name` and `description` fields, instead of the form that the acronym entrytypes use. This is necessary to get around the limitation that LaTeX cannot use mathematical markup within citation keys.

gloss.tex

```
\documentclass{article}

\newcommand{\citename}[1]{#1%
   \protect\nocite{#1}}

\makeatletter
\renewcommand\@biblabel[1]{#1}
\renewenvironment{thebibliography}[1]
{\section*{\refname}%
   \list{}{\setlength\labelwidth{1.5cm}%
      \leftmargin\labelwidth
      \advance\leftmargin\labelsep
      \let\makelabel\descriptionlabel}}%
{\endlist}
\makeatother

\renewcommand\refname{Glossary, List of %
Symbols, and Nomenclature}

\begin{document}

While \citename{Tilt} aberration changes only
the magnification, \citename{SA} induces
image blur. Wavefront aberration is a
function of radial distance \cite{sym:rho}
and azimuthal angle \cite{sym:phi}. Of the
two aberrations, only \citename{SA} has an
effect on the \citename{PSF} and
\citename{MTF}.

\bibliographystyle{gloss}
\bibliography{gloss}
\end{document}
```

gloss.bst

```
TEMPLATES:
symbol = <description>.
gloss = <description>.
acronym = <description>
SPECIAL-TEMPLATES:
sortkey = <citekey>
citelabel = <name>
OPTIONS:
bibitemsep = 0pt
replace_newlines = True
case_sensitive_field_names = True
```

gloss.bib

```
@symbol{sym:phi,
name = "$\phi$",
description = {Azimuthal angle.}}

@symbol{sym:rho,
name = "$\rho$",
description = {Radial distance from the
   optical axis.}}

@gloss{SA,
name = {Spherical Aberration},
description = {The departure from an ideal
   spherical wavefront that increases
   quadratically with radial distance.}}

@gloss{Tilt,
name = {Tilt aberration},
description = {A linear departure from an
   ideal wavefront --- equivalent to a
   magnification error.}}

@acronym{MTF = "Modulation Transfer
function"}
@acronym{PSF = "Point Spread Function"}
```

gloss.bbl

```
\begin{thebibliography}{6}
\setlength{\itemsep}{0pt}

\bibitem[MTF]{MTF}
Modulation Transfer function
\bibitem[PSF]{PSF}
Point Spread Function
\bibitem[Spherical Aberration]{SA}
The departure from an ideal spherical
wavefront that increases quadratically
with radial distance.
\bibitem[$\phi$]{sym:phi}
Azimuthal angle.
\bibitem[$\rho$]{sym:rho}
Radial distance from the optical axis.
\bibitem[Tilt aberration]{Tilt}
A linear departure from an ideal wavefront
--- equivalent to a magnification error.

\end{thebibliography}
```

Figure 5: The main `tex` file, style template (`bst`) file, bibliography database (`bib`) file, and resulting output (`bbl`) file for the glossary example.

Note that, because it is only a back-end engine, Bibulous by itself cannot split the glossary, list of acronyms, and list of symbols into three separate sections of the document. Doing so requires work by LaTeX itself, and thus requires changes to the *front* end. The example shown here provides only a minimal amount of front-end work — defining the `\citename{}` command, as well as redefining the `\biblabel` command and the `\thebibliography` environment — to make the bibliography structure appear as a list of definitions.

Glossary, List of Symbols, and Nomenclature

MTF Modulation Transfer function

PSF Point Spread Function

Spherical Aberration The departure from an ideal spherical wavefront that increases quadratically with radial distance.

ϕ Azimuthal angle.

ρ Radial distance from the optical axis.

Tilt aberration A linear departure from an ideal wavefront — equivalent to a magnification error.

Figure 6: The formatted glossary, list of symbols, and nomenclature (list of acronyms) for the inputs shown in Fig. 5. Note that the symbols ϕ and ρ are sorted by citation key, which are `sym:phi` and `sym:rho` for the case shown here.

Table 1: Timing comparison for BibTeX, Bibulous, and Biber, showing the amount of time required to generate a `.bbl` file for every entry in the given database.

	Database size (# citations)		
Tool	**100**	**820**	**12 000**
BibTeX	0.112 sec	0.117 sec	1.01 sec
Bibulous	0.190 sec	1.667 sec	41 sec
Biber	1.115 sec	9.051 sec	17.5 min

5 Comparison with existing bibliography engines

Bibulous not only provides easy customization for users, but does so with minimal compromises in speed and without requiring a compiler for installing. It is written in Python and has no external dependencies (it requires only Python's standard library), making it highly portable and easy to install, since Python is widely available on all modern platforms. And although it is slower than BibTeX, it is considerably faster than Biber.

To demonstrate processing speeds, each of the three back-end processors were run on three bibliography databases inside Bibulous' regression testing suite: one containing 12 000 entries, another with 820 entries, and a third with 100 entries. Using a standard desktop computer, running all three processors on these databases produces the timing results shown in Table 1. While the comparison is not entirely fair, since Biber is doing more work (especially by building hash codes for every author in every entry), the results show that Biber's expanded capabilities have come at a significant sacrifice in speed.

Biblatex and Biber are quickly replacing BibTeX as the standard tools of choice for LaTeX bibliography processors, as they provide much-needed functionality that BibTeX does not: support for larger databases and non-English languages, internationalization, localization, multilingualism, and more. In order to specify a bibliography style, Biblatex uses a combination of keyword options that are set within the main `.tex` file to specify bibliography formats. A weakness of this approach is that a user must know a large set of keywords and how to combine these to generate a custom style. Learning how to format references can require reading and understanding a substantial volume of Biblatex's documentation. With style templates, however, even first-time users can see how to customize the layout without needing to consult documentation, so that entirely new styles can be written up in a matter of minutes.

Beyond its use of templates, Bibulous has other important differences from BibTeX, including:

1. Bibulous converts LaTeX-markup accented characters to Unicode prior to performing sort functions. That is, while BibTeX will sort `t\^ete`, `t{\^e}te`, and `t{\^{e}}te` all as `tete` but `tête` separately as `tête`, Bibulous will sort all of these cases identically as `tête`. In addition, Bibulous automatically detects the user's default locale and sorts according to standard string collation algorithms.
2. Bibulous adds new options for formatting names inside `.bib` file entries, so that individual names are also allowed *three* or *four* comma delimiters. This gives users more freedom to control name layout.
3. Bibulous performs citation extraction by default — the relevant database entries from the main `.bib` files are extracted and placed together into a single (usually small) database. This speeds up program execution when operating with a large main database but a relatively

small number of citations.

4. Bibulous provides an `authorextract` command that creates a `.bib` file containing only those in which the given author's name is referenced. This can be useful for writing a CV.
5. Bibulous' `sentence_case` operator is almost identical to BibTeX's `change.case$` command, but works with Unicode.
6. Bibulous provides a Python API to its own internal data structures, allowing users to extend or customize the program as they desire, by placing function definitions within style files.

6 Comments

Rather than providing a detailed explanation of usage, the aim here has been to introduce Bibulous to LaTeX users and to showcase some of its strengths. These include:

1. using the simplicity and power of style templates to specify and customize bibliographies;
2. implementing a fully Unicode-compliant program without serious memory limitations.

The examples provided in the sections above show how these traits can be used to provide a powerful backend bibliography processor.

However, a range of functionality remains that is important for users but is not readily accessible through Bibulous alone. Tasks such as advanced citation labelling, the creation of indexes, etc. require integrating Bibulous with a LaTeX package front-end. Bibulous' `authorextract` functionality, for example, makes it easy to create a publications list (such as for a CV), but without front-end integration it has no means of separating reference entries into independent lists. That is, one would often like a `cvpublications.bst` style template that will sort references into sections differentiated by entry type (*e.g.* articles, book, inproceedings), with entries in each section sorted in reverse chronological order. Without integration with a front-end package, Bibulous can only provide a single list as output for LaTeX.

Similarly, advanced citation labelling, such as using citation commands like `natbib`'s `\citep{·}` and `\citet{·}`, require that additional information be written to the `.aux` file to inform the bibliography processor about the specific requirements of that specific reference's citation label. This kind of integration is a goal for version 2.0 of Bibulous.

Users interested in learning more details about how to use Bibulous can find much more in the user manual, available at the project website and in the source code repository. The source code for the entire repository (the core `bibulous.py` file plus testing software and documentation) can be found at `https://github.com/nzhagen/bibulous`.

References

[1] P. Lehman, A. Boruvka, P. Kime, and J. Wright, "The biblatex package: Programmable bibliographies and citations". Available from CTAN: `http://ctan.org/pkg/biblatex`.

[2] P. Kime and F. Charette, "biber: A backend bibliography processor for biblatex". Available from CTAN: `http://ctan.org/pkg/biber`.

[3] P. L. Kime, "Biber — the next generation backend processor for BibLaTeX". *TUGboat* **33**: 13–15 (2012).

[4] `http://tex.stackexchange.com/questions/28010/how-to-create-multilingual-english-japanese-bibliographies-with-biblatex-bib`.

◇ Nathan Hagen
Tucson AZ, USA
`nhagen (at) optics dot arizona dot edu`
`https://github.com/nzhagen/bibulous`