biber

A backend bibliography processor for biblatex

François Charette, Philip Kime
firmicus@ankabut.net,
Philip@kime.org.uk

Version biber 0.9.4 (biblatex 1.6)
28th July 2011

Contents

1 Introduction

1.1 About

`biber` is conceptually a `bibtex` replacement for `biblatex`. It is written in `Perl` with the aim of providing a customised and sophisticated data preparation backend for `biblatex`. You do *not* need to install `Perl` use `biber`—binaries are provided for many operating systems via the main TeX distributions (TeXLive, MacTeX, MiKTeX) and also via download from SourceForge. Functionally, `biber` offers a superset of `bibtex`'s capabilities but is tightly coupled with `biblatex` and cannot be used as a stand-alone tool with standard `.bst` styles. `biber`'s role is to support `biblatex` by performing the following tasks:

- Parsing data from data sources
- Processing cross-references, entry sets, related entries
- Generating data for name, name list and name/year disambiguation
- Structural validation according to `biblatex` data model
- Sorting reference lists
- Outputting data to a `.bbl` for `biblatex` to consume

1.2 Requirements

biber is distributed primarily as a stand-alone binary and is included in TeXLive, MacTeX and MiKTeX. If you are using any of these distributions, you do not need any additional software installed to use biber. You do *not* need a Perl installation at all to use the binary distribution of biber[1].

biber is developed on SourceForge[2] and this is the primary location for development releases, forums and bugfixes etc. It is included into TeXLive from the SourceForge releases.

1.3 License

biber is released under the free software Artistic License 2.0[3]

1.4 History

bibtex has been the default (only ...) integrated choice for bibliography processing in TeX for a long time. It has well known limitations which stem from its data format, data model and lack of Unicode support[4]. The .bst language for writing bibliography styles is painful to learn and use. It is not a general programming language and this makes it really very hard to do sophisticated automated processing of bibliographies.

biblatex was a major advance for LaTeX users as it moved much of the bibliography processing into LaTeX macros. However, biblatex still used bibtex as a sorting engine for the bibliography and also to generate various labels for entries. bibtex's capabilities even for this reduced set of tasks was still quite restricted due to the lack of Unicode support and the more and more complex programming issues involved in label preparation and file encoding.

biber was designed specifically for biblatex in order to provide a powerful backend engine which could deal with any required tasks to do with .bbl preparation. Its main features are:

- Deals with the full range of UTF-8
- Sorts in a completely customisable manner, using when available, CLDR collation tailorings
- Allows for per-entrytype options
- Automatically encodes the .bbl into any supported encoding format[5]

[1] If you prefer, you can run biber as a normal Perl program and doing this *does* require you to have a Perl interpreter installed. See section 3.

[2] http://sourceforge.net/projects/biblatex-biber/

[3] http://www.opensource.org/licenses/artistic-license-2.0.php

[4] In fact, there is now a Unicode version

[5] 'Supported' here means encodings supported by the Perl Encode module

- Processes all bibliography sections in one pass of the tool
- Handles UTF-8 citekeys and filenames (given a suitable fully UTF-8 compliant TeX engine)
- Creates entry sets dynamically and allow easily defined static entry sets, all processed in one pass
- Flexible user-customisable crossreference field inheritance model
- Handles complex auto-expansion and contraction of names and namelists[6]
- Support for related entries, to enable generic treatment of things like 'translated as', 'reprinted as', 'reprint of' etc.[7]
- Extensible modular data sources architecture for ease of adding more data source types.
- Support for remote data sources

1.5 Performance

`biber` can't really be compared with `bibtex` in any meaningful way performance-wise. `biber` is written in `Perl` and does a great deal more than `bibtex` which is written in C. One of `biber`'s test cases is a 2150 entry, 15,000 line `.bib` file which references a 630 entry macros file with a resulting 160 or so page (A4) formatted bibliography. This takes `biber` about 100 seconds on average to process on a reasonable computer. This is perfectly acceptable, especially for a batch program …

1.6 Acknowledgements

François Charette originally wrote `biber`. Philip Kime joined in the development in 2009.

2 Use

Firstly, please note that `biber` will *not* attempt to sanitise the content of `bibtex` data sources. That is, don't expect it to auto-escape any TeX special characters like '`&`' or '`%`' which it finds in, for example, your `TITLE` fields. It used to do this in earlier versions in some cases but as of version 0.9, it doesn't because it's fraught with problems and leads to inconsistent expectations and behaviour between different data source types. In your `bibtex` data sources, please make sure your entries are legal TeX code.

Running `biber --help` will display all options and a brief description of each. This is the most useful brief source of usage information. `biber` returns an exit code of 0 on success or 1 if there was an error.

[6] Coming in BibLaTeX 1.4
[7] Coming in BibLaTeX 1.5

Most biber options can be specified in long or short format. When mentioning options below, they are referred to as 'long form|short form' when an option has both a long and short form. As usual with such options, when the option requires an argument, the long form is followed by an equals sign '=' and then the argument, the short form is followed by a space and then the argument. For example, the --configfile|-g option can be given in two ways:

```
biber --configfile=somefile.conf
biber -g somefile.conf
```

With the backend=biber option, biblatex switches its backend interface and passes all options and information relevant to biber's operation in a control file with extension .bcf[8]. This is conceptually equivalent to the .aux file which LaTeX uses to pass information to bibtex. The .bcf file is XML and contains many options and settings which configure how biber is to process the bibliography and generate the .bbl file.

The usual way to call biber is simply with the .bcf file as the only argument. The '.bcf' extension of the control file is not optional. biblatex always outputs a control file with the .bcf extension. Specifying the '.bcf' extension to biber *is* optional. Assuming a control file called test.bcf, the following two commands are equivalent:

```
biber test.bcf
biber test
```

2.1 Options and config file

biber sets its options using the following resource chain which is given in decreasing precedence order:

command line options →
  biber.conf file →
    .bcf file→
      biber hard-coded defaults

Users do not need to care directly about the contents or format of the .bcf file as this is generated from the options which they specify for biblatex. The config file is a place to set commonly used command-line options and also to set options which cannot be set on the command line.

[8] BibLaTeX Control File

The configuration file is by default called `biber.conf` but this can be changed using the `--configfile|-g` option. Unless `--configfile|-g` is used, the config file is looked for in the following places, in decreasing order of preference:

`biber.conf` in the current directory →
  `$HOME/.biber.conf` →
    `$XDG_CONFIG_HOME/biber/biber.conf` →
      `$HOME/Library/biber/biber.conf` (Mac OSX only)
      `$APPDATA/biber.conf` (Windows only) →
        the output of '`kpsewhich biber.conf`' (if available on the system)

The config file format is a very flexible one which allows users to specify options in most common formats, even mixed in the same file. It's easier to see an example. Here is a config file which displays the `biber` defaults:

```
bblencoding         UTF-8
bibencoding         UTF-8
collate             1
<collate_options>
    level           3
</collate_options>
debug               0
fastsort            0
mincrossrefs        2
nolog               0
nostdmacros         0
<nosort>
  # ignore prefices like 'al-' when sorting name fields
  type_names      \A\p{L}{2}\p{Pd}
  # ignore diacritics when sorting author
  type_names      [\x{2bf}\x{2018}]
</nosort>
onlylog             0
quiet               0
sortcase            true
sortlocale          en_US.utf8
sortupper           true
trace               0
validate_control    0
validate_structure  0
wraplines           0
```

You can see here that options with multiple key/value pairs of their own like `--collate_options|-c` can be specified in Apache config format. Please see the documentation for the `Config::General Perl` module[9] if you really need details. In practise, if you use a config file at all for `biber`, it will contain very little as you will usually set all options by setting options in `biblatex` which will pass them to `biber` via the `.bcf` file.

The `--collate_options|-c` option takes a number of key/value pairs as value. See section 2.4 for details.

2.1.1 The 'map' option

The supplied data source drivers implement a default mapping from data source entrytypes and fields into the `biblatex` data model. If you want to override or augment the driver mappings you can use the `map` option which makes it possible to, for example, have a data source with non-standard entrytypes or fields and to have these automatically mapped into valid `biblatex` data types without modifying your data source.

The `map` option can only be set in the config file and not on the command line as it has a complex structure. This option allows you to perform various data source mapping (aliasing) tasks:

- Map data source entrytypes to `biblatex` entrytypes, optionally also adding new fields to the entry.
- Map data source fields to `biblatex` fields, optionally also adding new fields to the entry. This is basically a one→many field mapping. You can also limit field mappings to specific data source entrytypes.
- As a special case of the above, you can map data source fields to null, effectively removing them from the input stream.

The format of the `map` option in the config file is described below. Items in red are not literal, they are descriptive meta-values which are explained in the accompanying text. Items in blue are optional within their parent section. The entire `map` option is not case sensitive but the example below use uppercase in places for clarity. The general structure is:

```
<map>
  <driver1>
      ...
  </driver1>
      ⋮
  <drivern>
```

[9] http://search.cpan.org/search?query=Config::General&mode=all

```
      ...
  </drivern>
</map>
```

Here, driver1…drivern are the names of valid biber data source drivers (see section 2.9). Reserved tokens with special meanings for this option have the prefix 'BMAP_' and are described below. Each driver section contains one or more of the following subsections:

- Entrytype mapping
- Simple global field mapping
- Extended global field mapping
- Entrytype specific field mapping

Entrytype mapping

```
<entrytype source-entrytype>
  BMAP_TARGET target-entrytype
  <alsoset>
    target-field1 target-value1
       ⋮
    target-fieldn target-valuen
  </alsoset>
</entrytype>
```

An entrytype specification maps the data source entrytype source-entrytype into the biblatex entrytype target-entrytype. The literal string BMAP_TARGET is a constant and must be used to specify the target entrytype. The alsoset section is optional and specifies one or more target-fields to set in the entry, along with their target-values. The literal token BMAP_ORIGENTRYTYPE can be used as the target-value and it resolves to the data source entrytype name. For example:

```
<map>
  <bibtex>
    <entrytype CONVERSATION>
      BMAP_TARGET CUSTOMA
      <alsoset>
        USERA BMAP_ORIGENTRYTYPE
        NOTE "Auto-created this field"
      </alsoset>
    </entrytype>
    <entrytype RECORD>
      BMAP_TARGET MUSIC
    </entrytype>
  </bibtex>
</map>
```

This would turn a CONVERSATION entrytype in all bibtex data sources into a CUSTOMA entrytype and would automatically set the USERA field in the entry to 'conversation' and the NOTE field to 'Auto-created this field'. It also turns a RECORD entrytype into a MUSIC entrytype.

Simple global field mapping

```
<globalfield>
  source-field1 target-field1
                ⋮
  source-fieldn target-fieldn
</globalfield>
```

This section maps data source fields to biblatex fields, regardless of the entrytypes in which they occur (hence 'global'). You may use the special target-field constant BMAP_NULL to map a field to null, that is, to ignore it. This essentially has the effect of treating the field as if it did not exist in the data source. For example (to add to the previous example, so that you can see a more complete mapping example. The previously discussed example settings are greyed):

```
<map>
  <bibtex>
    <entrytype CONVERSATION>
      BMAP_TARGET CUSTOMA
      <alsoset>
        USERA BMAP_ORIGENTRYTYPE
        NOTE "Auto-created this field"
      </alsoset>
    </entrytype>
    <entrytype RECORD>
      BMAP_TARGET MUSIC
    </entrytype>
    <globalfield>
      ABSTRACT BMAP_NULL
      PARTICIPANT NAMEA
    </globalfield>
  </bibtex>
  <ris>
    N2 BMAP_NULL
  </ris>
</map>
```

Here, we have specified that we should ignore (map to BMAP_NULL) any ABSTRACT fields and that we should map all PARTICIPANT fields to the NAMEA field. We have

also specified that we should ignore the `N2` field in `ris` data sources (this is often used for abstracts in RIS).

Extended Global field mapping

```
<globalfield source-field>
  BMAP_TARGET target-field
  <alsoset>
    target-field1 target-value1
                  ⋮
    target-fieldn target-valuen
  </alsoset>
</globalfield>
```

Unlike the simple global mapping, this form maps only one `source-field`. You must specify the `target-field` using the constant `BMAP_TARGET`. The `alsoset` section is optional and specifies one or more `target-field`s to set in the entry, along with their `target-value`s. The literal token `BMAP_ORIGFIELD` can be used as the `target-value` and it resolves to the data source field name. Continuing our example configuration:

```
<map>
  <bibtex>
    <entrytype CONVERSATION>
      BMAP_TARGET CUSTOMA
      <alsoset>
        USERA BMAP_ORIGENTRYTYPE
        NOTE "Auto-created this field"
      </alsoset>
    </entrytype>
    <entrytype RECORD>
      BMAP_TARGET MUSIC
    </entrytype>
    <globalfield>
      ABSTRACT BMAP_NULL
      PARTICIPANT NAMEA
    </globalfield>
    <globalfield PUBMEDID>
       BMAP_TARGET EPRINT
      <alsoset>
        EPRINTTYPE BMAP_ORIGFIELD
        NOTE "Auto-created this field"
      </alsoset>
    </globalfield>
  </bibtex>
```

```
  <ris>
    N2 BMAP_NULL
  </ris>
</map>
```

Here, in all entries, the PUBMEDID field becomes the EPRINT field and the EPRINTTYPE field is set to 'pubmedid'. The NOTE field is set to 'Auto-created this field'.

Entrytype specific field mapping

```
<field source-field>
  BMAP_PERTYPE source-entrytype1
                ⋮
  BMAP_PERTYPE source-entrytypen
  BMAP_TARGET target-field
  <alsoset>
    target-field1 target-value1
                ⋮
    target-fieldn target-valuen
  </alsoset>
</field>
```

Here we map the data source source-field to the target-field but only if it occurs within one of the data source entrytypes source-entrytype1 … source-entrytypen. The source entrytype restrictions must be specified using the BMAP_PERTYPE token. The target field must be specified using the BMAP_TARGET token. The alsoset section is optional and specifies one or more target-fields to set in the entry, along with their target-values. The literal token BMAP_ORIGFIELD can be used as the target-value and it resolves to the data source field name. Continuing the example:

```
<map>
  <bibtex>
    <entrytype CONVERSATION>
      BMAP_TARGET CUSTOMA
      <alsoset>
        USERA BMAP_ORIGENTRYTYPE
        NOTE "Auto-created this field"
      </alsoset>
    </entrytype>
    <entrytype RECORD>
      BMAP_TARGET MUSIC
    </entrytype>
    <globalfield>
```

```
      ABSTRACT BMAP_NULL
      PARTICIPANT NAMEA
    </globalfield>
    <globalfield PUBMEDID>
       BMAP_TARGET EPRINT
      <alsoset>
       EPRINTTYPE BMAP_ORIGFIELD
       NOTE "Auto-created this field"
      </alsoset>
    </globalfield>
    <field NOTE>
      BMAP_PERTYPE ARTICLE
      BMAP_PERTYPE BOOK
      BMAP_TARGET BMAP_NULL
      <alsoset>
        ADDENDUM "A string"
      </alsoset>
    </field>
  </bibtex>
  <ris>
    N2 BMAP_NULL
  </ris>
</map>
```

Here we have specified that within data source entrytypes `ARTICLE` and `BOOK`, the field `NOTE` should be ignored and the `ADDENDUM` field should be set to 'A string'.

Please note that the data source entrytypes and fields are *not* necessarily the same as in the `biblatex` data model. The definitive guide to the `biblatex` data model is the `biblatex` documentation. It is certainly the case that for `bibtex` data sources, there is a large degree of congruence between the data source entrytypes and fields and the `biblatex` data model because `biblatex` grew out of `bibtex`. However, for other data sources like `ris`, `endnotexml` and `zoterordfxml`, the source entrytypes and fields are usually very differently modelled and named. For example, here is how to drop the 'subject' fields from various entrytypes in Zotero XML RDF format data sources:

```
<map>
  <zoterordfxml>
    <field "dc:subject">
      BMAP_PERTYPE journalArticle
      BMAP_PERTYPE book
      BMAP_PERTYPE bookSection
      BMAP_TARGET BMAP_NULL
    </field>
  </zoterordfxml>
</map>
```

Or here, mapping journal articles into REPORT entries for Endnote XML format data sources.

```
<map>
  <endnotexml>
    <entrytype "journal Article">
      BMAP_TARGET REPORT
    </entrytype>
  </endnotexml>
</map>
```

2.1.2 The 'nosort' option

The value of the nosort option can only be set in the config file and not on the command line. This is because the values are Perl regular expressions and would need special quoting to set on the command line. This can get a bit tricky on some OSes (like Windows) so it's safer to set them in the config file. In any case, it's unlikely you would want to set them for particular biber runs; they would more likely be set as your personal default and thus they would naturally be set in the config file anyway. nosort allows you to ignore parts of a field for sorting. This is done using Perl regular expressions which specify what to ignore in a field. You can specify as many patterns as you like for a specific field. Also available are some field type aliases so you can, for example, specify patterns for all name fields or all title fields. These field types all begin with the string 'type_', see Table 1.

For example, this option can be used to ignore diacritic marks and prefices in names which should not be considered when sorting. Given (the default):

```
<nosort>
  type_names       \A\p{L}{2}\p{Pd}
  type_names       [\x{2bf}\x{2018}]
</nosort>
```

and the .bib data source entry:

```
author     = {{al-Hasan}, ᶜAlī},
```

the prefix 'al-' and the diacritic 'ᶜ' will not be considered when sorting. See the Perl regular expression manual page for details of the regular expression syntax[10].

If a nosort option is found for a specific field, it will override any option for a type which also covers that field.

Here is another example. Suppose you wanted to ignore 'The' at the beginning of a TITLE field when sorting, you could add this to your biber.conf:

[10] http://perldoc.perl.org/perlre.html

Alias	Fields
type_name	author
	afterword
	annotator
	bookauthor
	commentator
	editor
	editora
	editorb
	editorc
	foreword
	holder
	introduction
	namea
	nameb
	namec
	shortauthor
	shorteditor
	translator
type_title	booktitle
	eventtitle
	issuetitle
	journaltitle
	maintitle
	origtitle
	title

Table 1: nosort option field type aliases

```
<nosort>
  title       \AThe\s+
</nosort>
```

If you wanted to do this for all title fields listed in Table 1, then you would do this:

```
<nosort>
  type_title       \AThe\s+
</nosort>
```

Note: `nosort` can be specified for most fields but not for things like dates and special fields as that wouldn't make much sense.

2.2 Input/Output File Locations

2.2.1 Control file

The control file is normally passed as the only argument to biber. It is searched for in the following locations, in decreasing order of priority:

Absolute filename →
  In the `--output_directory`, if specified→
    Relative to current directory→
      Using `kpsewhich`, if available

2.2.2 Data sources

Local data sources of type 'file' are searched for in the following locations, in decreasing order of priority:

Absolute filename →
  In the `--output_directory`, if specified→
    Relative to current directory→
      In the same directory as the control file→
        Using `kpsewhich` for supported formats, if available

Remote file data sources (beginning with `http://` or `ftp://`) are retrieved to a temp file and processed as normal. Users do not specify explicitly the bibliography database files; they are passed in the `.bcf` control file, which is constructed from the `biblatex` `\addbibresource{}` macros.

2.3 Logfile

By default, the logfile for biber will be named `\jobname.blg`, so, if you run

```
biber <options> test.bcf
```

then the logfile will be called '`test.blg`'. Like the `.bbl` output file, it will be created in the `--output_directory|-c`, if this option is defined. You can override the logfile name by using the `--logfile` option:

```
biber --logfile=lfname test.bcf
```

results in a logfile called '`lfname.blg`'.

Warning: be careful if you are expecting `biber` to write to directories which you don't have appropriate permissions to. This is more commonly an issue on non-Windows OSes. For example, if you rely on `kpsewhich` to find your database files which are in system TeX directories, you may well not have write permission there so `biber` will not be able to write the `.bbl`. Use the `--outfile|-O` option to specify the location to write the `.bbl` to in such cases.

2.4 Collation and Localisation

`biber` takes care of collating the bibliography for `biblatex`. It writes entries to the `.bbl` file sorted by a completely customisable set of rules which are passed in the `.bcf` file by `biblatex`. `biber` has two ways of performing collation:

`--collate|-C`
: The default. This option makes `biber` use the `Unicode::Collate` module for collation which implements the full UCA (Unicode Collation Algorithm). It also has CLDR (Common Locale Data Repository) tailoring to deal with cases which are not covered by the UCA. It is a little slower than `--fastsort|-f` but the advantages are such that it's rarely worth using `--fastsort|-f`

`--fastsort|-f`
: Biber will sort using the OS locale collation tables. The drawback for this method is that special collation tailoring for various languages are not implemented in the collation tables for many OSes. For example, few OSes correctly sort 'å' before 'ä' in the Swedish (`sv_SE`) locale. If you are using a common latin alphabet, then this is probably not a problem for you.

The locale used for collation is determined by the following resource chain which is given in decreasing precedence order:

```
--collate_options|-c (e.g. -c 'locale => "de_DE"') →
  --sortlocale|-l →
    LC_COLLATE environment variable →
      LANG environment variable →
        LC_ALL environment variable
```

With the default `--collate|-C` option, the locale will be used to look for a collation tailoring for that locale. It will generate an informational warning if it finds none. This is not a problem as the vast majority of collation cases are covered by the standard UCA and many locales neither have nor need any special collation tailoring.

With the `--fastsort|-f` option, the locale will be used to locate an OS locale definition to use for the collation. This may or may not be correctly tailored, depending on the locale and the OS.

Collation is by default case sensitive. You can turn this off globally using the `biber` option `--sortcase=false` or from `biblatex` using its option `sortcase=false`. The option can also be defined per-field so you can sort some fields case sensitively and others case insensitively. See the `biblatex` manual.

`--collate|-C` by default collates uppercase before lower. You can reverse this globally for all sorting using the `biber` option `--sortupper=false` or from `biblatex` by using its option `sortupper=false`. The option can also be defined per-field so you can sort some fields uppercase before lower and others lower before upper. See the `biblatex` manual. Be aware though that some locales rightly enforce a particular setting for this (for example, Danish). You will be able to override it but `biber` will warn you if you do. `sortupper` has no effect when using `--fastsort|-f`–you are at the mercy of what your OS locale does.

There are in fact many options to `Unicode::Collate` which can tailor the collation in various ways in addition to the locale tailoring which is automatically performed. Users should see the the documentation to the module for the various options, most of which the vast majority of users will never need[11]. Options are

[11] For details on the various options, see http://search.cpan.org/search?query=Unicode%3A%3ACollate&mode=all

passed using the `--collate_options|-c` option as a single quoted string, each option separated by comma, each key and value separated by '=>'. See examples.

2.4.1 Examples

`biber`
: Call biber using all settings from the `.bcf` generated from the LaTeX run. Case sensitive UCA sorting is performed taking the locale for tailoring from the environment if no `sortlocale` is defined in the `.bcf`

`biber --sortlocale=de_DE`
: Override any locale setting in the `.bcf` or the environment.

`biber --fastsort`
: Use slightly quicker internal sorting routine. This uses the OS locale files which may or may not be accurate.

`biber --sortcase=false`
: Case insensitive sorting.

`biber --sortupper=false --collate_options="backwards => 2"`
: Collate lowercase before upper and collate French accents in reverse order at UCA level 2.

2.5 Encoding of files

`biber` takes care of reencoding the data source data as necessary. In normal use, `biblatex` passes its `bibencoding` option value to `biber` via the `.bcf` file. It also passes the value of its `texencoding` option (which maps to `biber`'s `bblencoding|-E` option) the default value of which depends on which TeX engine and encoding packages you are using (see `biblatex` manual for details).
`biber` performs the following tasks:

1. Decodes the data source into UTF-8 if it is not UTF-8 already
2. Decodes LaTeX character macros into UTF-8 if `--bblencoding|-E` is UTF-8
3. Encodes the output so that the `.bbl` is in the encoding that `--bblencoding|-E` specifies
4. Warns if it is asked to output to the `.bbl` any UTF-8 decoded LaTeX character macros which are not in the `--bblencoding|-E` encoding. Replaces with a suitable LaTeX macro

Normally, you do not need to set the encoding options on the `biber` command line as they are passed in the `.bcf` via the information in your `biblatex` environment. However, you can override the `.bcf` settings with the command line. The resource chain for encoding settings is, in decreasing order of preference:

--bibencoding|-e and --bblencoding|-E →
  biber config file →
    .bcf control file

2.5.1 LaTeX macro decoding

As mentioned above, biber sometimes converts LaTeX character macros into UTF-8. In fact there are two situations in which this occurs.

1. When --bblencoding|-E is UTF-8
2. Always for internal sorting purposes

This decoding is very useful but take note of the following two scenarios, which relate to each of the two situations in which LaTeX macro decoding occurs:

Decoding when output is UTF-8

If you are using PDFLaTeX and \usepackage[utf8]{inputenc}, it is possible that the UTF-8 characters resulting from biber's internal LaTeX character macro decoding break inputenc. This is because inputenc does not implement all of UTF-8, only a commonly used subset.

An example–if you had \DJ in your .bib data source, biber decodes this correctly to 'Đ' and this breaks inputenc because it doesn't understand that UTF-8 character. The real solution here is to switch to a TeX engine with full UTF-8 support like XeTeX or LuaTeX as these don't use or need inputenc. However, you can also try the --bblsafechars option which will try to convert any UTF-8 chars into LaTeX macros on output. The biblatex option 'texencoding=ascii' (which corresponds to the biber option '--bblencoding|-E') will automatically set --bblsafechars.

See also the biber --help output for the --bblsafecharsset option which can customise the set of conversion characters to use.

Decoding for internal sorting

If your bblencoding is not UTF-8, and you are using some UTF-8 equivalent LaTeX character macros in your .bib data source, then some .bbl fields (currently only \sortinit{}) might end up with invalid characters in them, according to the .bbl encoding. This is because some fields must be generated from the final sorting data which is only available after the LaTeX character macro decoding step.

For example, suppose you are using PDFLaTeX with \usepackage[latin1]{inputenc} and the following bibtex data source entry:

```
@BOOK{citekey1,
  AUTHOR = {{\v S}imple, Simon},
}
```

With normal LaTeX character macro decoding, the `{\v S}` is decoded into 'Š' and so with name-first sorting, `\sortinit{}` would be 'Š'. This is an invalid character in latin1 encoding and so the `.bbl` would be broken. In such cases when `\sortinit{}` is a char not valid in the `bblencoding`, `biber` tries to replace the character with a suitable LaTeX macro. The solution is really to use UTF-8 `.bbl` encoding whenever possible. In extreme cases where even with UTF-8 encoding, the char is not recognised by LaTeX due to an incomplete UTF-8 implementation (as with `inputenc`), this might also mean switching TeX engines to one that supports full UTF-8.

2.5.2 Examples

`biber`
: Set `bibencoding` and `bblencoding` from the config file or `.bcf`.

`biber --bblencoding=latin2`
: Encode the `.bbl` as latin2, overriding the `.bcf`.

`biber --bblsafechars`
: Set `bibencoding` and `bblencoding` from the config file or `.bcf`. Force encoding of UTF-8 chars to LaTeX macros using default conversion set.

`biber --bblencoding=ascii`
: Encode the `.bbl` as ascii, overriding the `.bcf`. Automatically sets `--bblsafechars` to force UTF-8 to LaTeX macro conversion.

`biber --bblencoding=ascii --bblsafecharsset=full`
: Encode the `.bbl` as ascii, overriding the `.bcf`. Automatically sets `--bblsafechars` to force UTF-8 to LaTeX macro conversion using the full set of conversions

`biber --decodecharsset=full`
: Set `bibencoding` and `bblencoding` from the config file or `.bcf`. Use the full LaTeX macro to UTF-8 conversion set because you have some more obscure character macros in your `.bib` data source which you want to sort correctly

`biber -u`
: Shortcut alias for `biber --bibencoding=UTF-8`

`biber -U`
: Shortcut alias for `biber --bblencoding=UTF-8`

2.6 Limitations

Currently, users are restricted to a one-one mapping from datasource entry types/fields to the `biblatex` supported entry type/fields. This is mitigated a little by the type/field aliases which `biblatex` supports. In the future, users will be able to customise the data source driver config in order to define their own entry type/field aliases so that there is more flexibility in mapping data source entry type/fields to internal `biblatex` types/fields.

2.7 Editor Integration

Here is some information on how to integrate `biber` into some of the more common editors

2.7.1 Emacs

Emacs has the very powerful AUcTeX mode for editing TeX and running compilations. BibTeX is already integrated into AUCTeX and it is quite simple to add support for `biber`. Use the Emacs Customise interface to modify the `TeX-command-list` variable and add a `Biber` command.

```
M-x customise-variable
TeX-command-list
```

and then `Ins` somewhere a new command that looks like Figure 1.

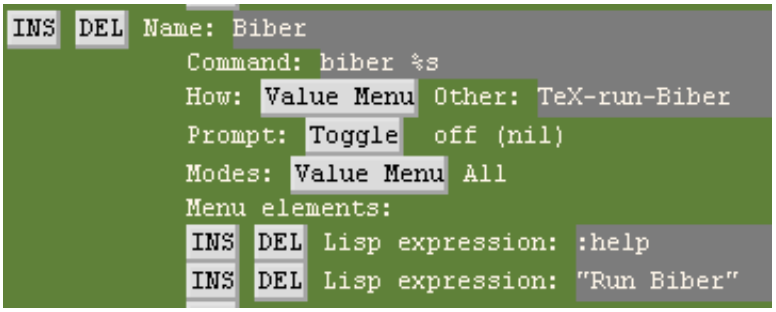

Figure 1: Screenshot of AUCTeX command setup for Biber

Alternatively, you can add it directly in lisp to your `.emacs` like this:

```
(add-to-list 'TeX-command-list
  (quote
    ("Biber" "biber %s" TeX-run-Biber nil t :help "Run Biber")))
```

However you add the command to `TeX-command-list`, customise the actual `Biber` command parameters as you want them, using '`%s`' as the LaTeX file name place holder. Then define the following two functions in your `.emacs`.

```
(eval-after-load "tex"
  (quote (defun TeX-run-Biber (name command file)
    "Create a process for NAME using COMMAND to format FILE with Biber."
    (let ((process (TeX-run-command name command file)))
      (setq TeX-sentinel-function 'TeX-Biber-sentinel)
      (if TeX-process-asynchronous
```

```
          process
        (TeX-synchronous-sentinel name file process))))
         )
  )

(eval-after-load "tex"
  (quote (defun TeX-Biber-sentinel (process name)
    "Cleanup TeX output buffer after running Biber."
    (goto-char (point-max))
    (cond
     ;; Check whether Biber reports any warnings or errors.
     ((re-search-backward (concat
                           "^(There \\(?:was\\|were\\) \\([0-9]+\\) "
                           "\\(warnings?\\|error messages?\\))") nil t)
      ;; Tell the user their number so that she sees whether the
      ;; situation is getting better or worse.
      (message (concat "Biber finished with %s %s. "
                       "Type `%s' to display output.")
               (match-string 1) (match-string 2)
               (substitute-command-keys
                "\\<TeX-mode-map>\\[TeX-recenter-output-buffer]")))
     (t
      (message (concat "Biber finished successfully. "
                       "Run LaTeX again to get citations right."))))
    (setq TeX-command-next TeX-command-default))
        )
  )
```

You'll then see a `Biber` option in your AUCTeX command menu or you can just `C-c C-c` and type `Biber`.

2.7.2 TeXworks

It's very easy to add `biber` support to TeXworks. In the Preferences, select the Typesetting tab and then add a new Processing Tool as in Figure 2.

2.8 BibTeX macros and the MONTH field

`BibTeX` defines macros for month abbreviations like '`jan`', '`feb`' etc. `biber` also does this, defining them as numbers since that is what `biblatex` wants. In case you are also defining these yourself (although if you are only using `biblatex`, there isn't much point), you will get macro redefinition warnings from the `btparse` library. You can turn off `biber`'s macro definitions to avoid this by using the option `--nostdmacros`.

`biber` will look at any `MONTH` field in a `bibtex` data source and if it's not a number as `biblatex` expects (because it wasn't one of the macros as mentioned

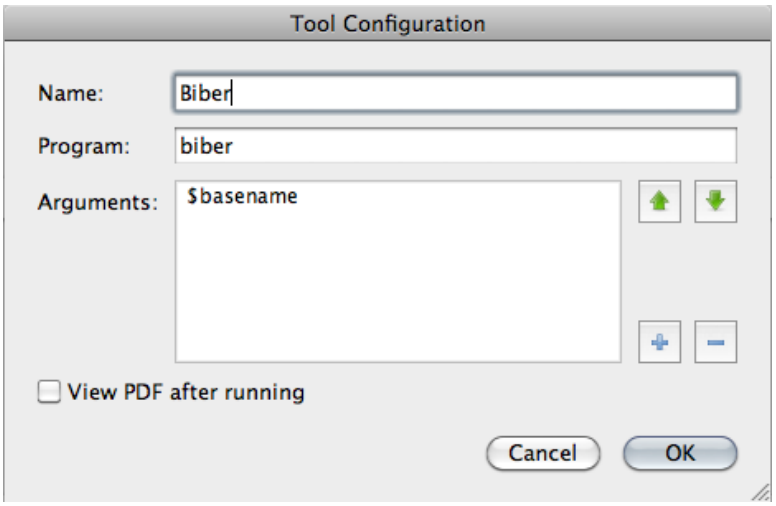

Figure 2: Screenshot of TeXworks processing tool setup for Biber

above or these macros were disabled by `--nostdmacros`), it will try to turn it into the right number in the `.bbl`. If you only use `biblatex` with your `bibtex` data source files, you should probably make any `MONTH` fields be the numbers which `biblatex` expects.

2.9 Biber data source drivers

`biber` uses a modular data source driver model to provide access to supported data sources. The drivers are responsible for mapping driver entry types and fields to the `biblatex` model. This is aided by a driver configuration file (`.dcf`). The information in this file for each driver can be found in the driver documentation folder on SourceForge[12]. This file shows you which handlers the driver uses for different fields and what certain entry types and fields are aliased to in the `biblatex` data model. This is not fantastically useful to know without knowing also what the named driver handlers do specifically but there is a 'description' section which mentions any special handling and general comments on the driver. You should read the documentation for the drivers you use to get an idea of how `biber` handles your data. Data model mapping is an imprecise art and the drivers are where the necessarily messy parts of `biber` live. Most data source models are not designed with typesetting in mind and are usually not fine-grained enough to provide the sorts of information that `biblatex` needs. `biber` does its best to obtain as much

[12] https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/current/documentation/drivers

meaningful information from a data source as possible. Currently supported data sources drivers are:

- `bibtex` — BibTeX data files
- `endnotexml` — Endnote XML export format, version $\geq$ Endnote X1
- `ris` — Research Information Systems format
- `zoterordfxml` — Zotero RDF XML format, version 2.0.9

3 Binaries

`biber` is a Perl application which relies heavily on quite a few modules. It is packaged as a stand-alone binary using the excellent `PAR::Packer` module which can pack an entire `Perl` tree plus dependencies into one file which acts as a stand-alone binary and is indistinguishable from such to the end user. You can also simply download the `Perl` source and run it as a normal `Perl` program which requires you to have a working `Perl` 5.14 installation and the ability to install the pre-requisite modules. You would typically only do this if you wanted to keep up with all the bleeding-edge git commits before they had been packaged as a binary. Almost all users will not want to do this and should use the binaries from their TeX distribution or downloaded directly from SourceForge in case they need to use a more recent binary than is included in their TeX distribution.

The binary distributions of biber are made using the `Perl PAR::Packer` module. They can be used as a normal binary but have some behaviour which is worth noting:

- Don't be worried by the size of the binaries. `PAR::Packer` essentially constructs a self-extracting archive which unpacks the needed files first.
- On the first run of a new version (that is, with a specific hash), they actually unpack themselves to a temporary location which varies by operating system. This unpacking can take a little while and only happens on the first run of a new version. **Please don't kill the process if it seems to take some time to do anything on the first run of a new binary**. If you do, it will not unpack everything and it will almost certainly break `biber`. You will then have to delete your binary cache (see section 3.1 below) and re-run the `biber` executable again for the first time to allow it to unpack properly.

3.1 Binary Caches

`PAR::Packer` works by unpacking the required files to a cache location. It only does this on the first run of a binary by computing a hash of the binary and comparing it with the cache directory name which contains the hash. So, if you run several versions of a binary, you will end up with several cached trees which are

never used. This is particularly true if you are regularly testing new versions of the `biber` binary. It is a good idea to delete the caches for older binaries as they are not needed and can take up a fair bit of space. The caches are located in a temporary location which varies from OS to OS. The cache name is:

`par-<username>/cache-<hash>` (Linux/Unix/OSX)
`par-<username>\cache-<hash>` (Windows)

The temp location is not always obvious but these are sensible places to look (where `*` can vary depending on username:

- `/var/folders/*/*/-Tmp-/` (OSX, local GUI login shell)
- `/var/tmp/` (OSX (remote ssh login shell), Unix)
- `/tmp/` (Linux)
- `C:\Documents and Settings\<username>\Local Settings\Temp` (Windows/Cygwin)
- `C:\Windows\Temp` (Windows)

To clean up, you can just remove the whole `par-<username>` directory/folder and then run the current binary again.

3.2 Binary Architectures

Binaries are available for many architectures, directly on SourceForge and also via TeXLive:

- `linux_x86_32`
- `linux_x86_64`
- `MSWin32`
- `cygwin32`
- `darwin_x86_64`
- `darwin_x86_i386`
- `freebsd_x86`[13]
- `freebsd_amd64`[13]
- `solaris_x86`[13]

If you want to run development versions, they are usually only regularly updated for the core architectures which are not flagged as third-party built above. If you want to regularly run the latest development version, you should probably git clone the relevant branch and run `biber` as a pure perl program directly.

[13] Binary maintained by third party. See README in binary download directory for this platform for support/contact details. Usually, the binary maintainer is also the binary build provider for TexLive.

3.3 Installing

These instructions only apply to manually downloaded binaries. If `biber` came with your TeX distribution just use it as normal.

Download the binary appropriate to you OS/arch[14]. Below I assume it's on your desktop.

You have to move the binary to somewhere in you command-line or TeX utility path so that it can be found. If you know how to do this, just ignore the rest of this section which contains some instructions for users who are not sure about this.

3.3.1 OSX

If you are using the TexLive MacTeX distribution:

```
sudo mv ~/Desktop/biber /usr/texbin/
sudo chmod +x /usr/texbin/biber
```

If you are using the macports TexLive distribution:

```
sudo mv ~/Desktop/biber /opt/local/bin/
sudo chmod +x /opt/local/bin/biber
```

The '`sudo`' commands will prompt you for your password.

3.3.2 Windows

The easiest way is to just move the executable into your `C:\Windows` directory since that is always in your path. A more elegant is to put it somewhere in your TeX distribution that is already in your path. For example if you are using MiKTeX:

```
C:\Program Files\MiKTeX 2.8\miktex\bin\
```

3.3.3 Unix/Linux

```
sudo mv ~/Desktop/biber /usr/local/bin/biber
sudo chmod +x /usr/local/bin/biber
```

Make sure `/usr/local/bin` is in your PATH. Search Google for 'set PATH linux' if unsure about this. There are many pages about this, for example: `http://www.cyberciti.biz/faq/unix-linux-adding-path/`

[14] `https://sourceforge.net/projects/biblatex-biber`

3.4 Building

Instructions for those who want/need to build an executable from the `Perl` version. For this, you will need to have `Perl` 5.14 with the following modules:

- All `biber` pre-requisites
- `PAR::Packer` and all dependencies

You should have the latest CPAN versions of all required modules as `biber` is very specific in some cases about module versions and depends on recent fixes in many cases. You can see if you have the `biber Perl` dependencies by the usual

```
perl ./Build.PL
```

invocation in the `biber Perl` distribution tree directory. Normally, the build procedure for the binaries is as follows[15]:

- Get the biber source tree from SF and put it on the architecture you are building for
- cd to the root of the source tree
- `perl Build.PL` (this will check your module dependencies)
- `Build test`
- `Build install` (may need to run this as sudo on UNIXesque systems)
- `cd dist/<arch>`
- `build.sh` (`build.bat` on Windows)

This leaves a binary called '`biber-<arch>`' (also with a '`.exe`' extension on Windows/Cygwin) in your current directory. The tricky part is constructing the information for the build script. There are two things that need to be configured, both of which are required by the `PAR::Packer` module:

1. A list of modules/libraries to include in the binary which are not automatically detected by the `PAR::Packer` dependency scanner
2. A list of extra files to include in the binary which are not automatically detected by the `PAR::Packer` dependency scanner

To build `biber` for a new architecture you need to define these two things as part of constructing new build scripts:

- Make a new subfolder in the `dist` directory named after the architecture you are building for. This name is arbitrary but should be fairly obvious like '`solaris-sparc-64`', for example.

[15] On UNIXequse systems, you may need to specify a full path to the scripts e.g. `./Build`

- Copy the `biber.files` file from an existing build architecture into this directory.
- For all of the files with absolute pathnames in there (that is, ones we are not pulling from the `biber` tree itself), locate these files in your `Perl` installation tree and put the correct path in the file.
- Copy the build script from a vaguely similar architecture (i.e. Windows/non-Windows …) to your new architecture directory.
- Change the `--link` options to point to where the required libraries reside on your system.
- Change the `--output` option to name the resulting binary for your architecture.
- Run the build script

The `--link` options can be a little tricky sometimes. It is usually best to build without them once and then run `ldd` (or OS equivalent) on the binary to see which version/location of a library you should link to. You can also try just running the binary and it should complain about missing libraries and where it expected to find them. Put this path into the `--link` option. The `--module` options are the same for all architectures and do not need to be modified. On architectures which have or can have case-insensitive file systems, you should use the build script from either Windows or OSX as a reference as these include a step to copy the main `biber` script to a new name before packing the binary. This is required as otherwise a spurious error is reported to the user on first run of the binary due to a name collision when it unpacks itself.

See the `PAR` wiki page[16] for FAQs and help on building with `PAR::Packer`. Take special note of the FAQs on including libraries with the packed binary[17].

3.4.1 Testing a binary build

You can test a binary that you have created by copying it to a machine which preferably doesn't have `perl` at all on it. Running the binary with no arguments will unpack it in the background and display the help. To really test it without having LaTeX available, get the two quick test files from SourceForge[18], put them in a directory and run `biber` in that directory like this:

```
biber --convert_control test
```

[16] http://par.perl.org/wiki/Main_Page

[17] http://par.perl.org/wiki/FAQ, section entitled 'My PAR executable needs some dynamic libraries'

[18] https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/testfiles

This will run `biber` normally on the test files plus it will also perform an XSLT transform on the `.bcf` and leave an HTML representation of it in the same directory thus testing the links to the XML and XSLT libraries as well as the `bibtex` parsing libraries. The output should look something like this (may be differences of `biber` version and locale of course but there should be no errors or warnings).

```
INFO - This is biber 0.9
INFO - Logfile is 'test.blg'
INFO - Converted BibLaTeX control file 'test.bcf' to 'test.bcf.html'
INFO - Reading test.bcf
INFO - Using all citekeys in bib section 0
INFO - Processing bib section 0
INFO - Processing bibtex format file 'test.bib' for section 0
INFO - Decoding LaTeX character macros into UTF-8
INFO - Sorting list 'MAIN' keys
INFO - No sort tailoring available for locale 'en_GB.UTF-8'
INFO - Sorting list 'SHORTHANDS' keys
INFO - No sort tailoring available for locale 'en_GB.UTF-8'
INFO - Writing 'test.bbl' with encoding 'UTF-8'
INFO - Output to test.bbl
```

There should now be these new files in the directory:

```
test.bcf.html
test.blg
test.bbl
```