# biber

**A backend bibliography processor for biblatex**

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

**Version biber 0.9.8 (biblatex 1.7)**
**15th December 2011**

## Contents



## 1 Important Changes

Please see the `Changes` file which accompanies Biber for the details on changes in each version. This section is just for important things like incompatible changes which users should be aware of.

### 0.9.7

The user config file has a completely new format.The reason for this is that the older `Config::General` format could not be extended to deal with more sophisticated features like per-datasource restrictions. An XML format is much better and in fact easier to understand. The old format of the `map` option (now called `sourcemap`) was rather confusing because of limitations in the old config file format. Please see section 3.1.1 and convert your config files to the new format.

### 0.9.6

Matching of citation keys and datasource entry keys is now *case-sensitive*. This is to enforce consistency across the entire BibLATEX and Biber processing chain. All of the

usual referencing mechanisms in LaTeX are case-sensitive and so is the matching in BibLaTeX of citations to entries in the `.bbl` file generated by Biber. It is inconsistent and messy to enforce case-insensitivity in only Biber's matching of citations keys to datasource entry keys. If Biber detects what looks like a case mismatch between citation keys, it will warn you.
Summary of warnings/errors is now a new format. When Biber finishes writing the `.bbl`, it gives a summary count of errors/warnings. It used to do this in the same format as BibTeX, for compatibility. Now it uses a more consistent and easier to parse format which matches all other Biber messages. Please note if you need to support Biber in an external tool. I have updated the notes on AUCTeX support below to reflect this.

# 2 Introduction

## 2.1 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

## 2.2 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].

[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 4.

| Biber version | BibLaTeX version |
|---|---|
| 0.9.8 | 1.7x |
| 0.9.7 | 1.7x |
| 0.9.6 | 1.7x |
| 0.9.5 | 1.6x |
| 0.9.4 | 1.5x |
| 0.9.3 | 1.5x |
| 0.9.2 | 1.4x |
| 0.9.1 | 1.4x |
| 0.9 | 1.4x |

Table 1: Biber/BibLaTeX compatibility matrix

Biber's git repository is on github[2]. Biber's documentation, binary downloads and supporting files are on SourceForge[3] and this is the primary location for development releases, forums and bugfixes etc. Biber is included into TeXLive, the binaries coming from SourceForge.

## 2.3 Compatibility Matrix

Biber versions are closed coupled with BibLaTeX versions. You need to have the right combination of the two. Biber will warn you during processing if it encounters information which comes from the wrong BibLaTeX version. Table 1 shows a compatibility matrix for the recent versions.

## 2.4 License

Biber is released under the free software Artistic License 2.0[4]

## 2.5 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[5]. 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.

[2] https://github.com/plk/biber
[3] http://sourceforge.net/projects/biblatex-biber/
[4] http://www.opensource.org/licenses/artistic-license-2.0.php
[5] In fact, there is now a Unicode version

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 tailoring
- Allows for per-entrytype options
- Automatically encodes the `.bbl` into any supported encoding format[6]
- Processes all bibliography sections in one pass of the tool
- Output to GraphViz instead of `.bbl` in order to help visualise complex bibliographies with many crossrefs etc. (see section 3.9)
- Handles UTF-8 citekeys and filenames (given a suitable fully UTF-8 compliant TeX engine)
- Creates entry sets dynamically and allows easily defined static entry sets, all processed in one pass
- 'Syntactic' inheritance via new `XDATA` entrytype and field. This can be thought of as a field-based generalisation of the BibTeX `@STRING` functionality (which is also supported).
- 'Semantic' inheritance via a generalisation of the BibTeX crossreference mechanism. This is highly customisable by the user—it is possible to choose which fields to inherit for which entrytypes and to inherit fields under different names etc. Nested crossreferences are also supported.
- Handles complex auto-expansion and contraction of names and namelists (See section 4.11.4 of the BibLaTeX manual for an excellent explanation with examples, this is quite an impressive feature …)
- Extensible modular data sources architecture for ease of adding more data source types
- Support for remote data sources
- User-definable mapping and suppression of fields and entrytypes in data sources. You can use this to, for example, ignore all `ABSTRACT` fields completely. See section 3.1.1
- Support for related entries, to enable generic treatment of things like 'translated as', 'reprinted as', 'reprint of' etc. (BibLaTeX support coming in BibLaTeX 2.x)

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

- Customisable labels (BibLaTeX support coming in BibLaTeX 2.x)
- Multiple bibliography lists in the same section with different sorting and filtering (BibLaTeX support coming in BibLaTeX 2.x)
- No more restriction to a static data model of specific fields and entrytypes. (BibLaTeX support coming in BibLaTeX 2.x)
- Structural validation of the data against the data model with a customisable validation model (BibLaTeX support coming in BibLaTeX 2.x)

Figure 1 shows the main functional units of processing in Biber. The most difficult tasks which Biber performs are the processing of BibLaTeX's `uniquename` and `uniquelist` options[7], the sorting of lists[8] and the initial data parse and remap into an internal data model. Biber is getting on for around 20,000 lines of mostly OO Perl and relies on certain splendid Perl modules such as `Unicode::Collate`, `Text::BibTeX` and `XML::LibXML`.

It may be useful to know something about the different routes a datasource entry can take as it passes through Biber.

1. All cited entries which are subsequently found in a datasource are instantiated in the internal Biber data model.
2. Some uncited entries on which cited entries depend are instantiated in the internal Biber data model:
   - Entries with entrytype `XDATA` which are referenced from cited entries.
   - Entries mentioned in the `CROSSREF` or `XREF` field of a cited entry (unless they are also cited themselves in which case they are already instantiated as per item 1 above).
   - Clones of entries mentioned as a 'related' entry of a cited entry[9].
   - Members of sets, either explicit `SET` entrytype entries or dynamic sets.
3. Some uncited but instantiated entries are promoted to cited status so that they make it into the output:
   - Entries instantiated by being members of a set.
   - Entries instantiated by being mentioned as a `CROSSREF` are promoted to cited status if `CROSSREF`'ed or `XREF`'ed at least `mincrosref` times.
   - Clones of entries mentioned as a 'related' entry of a cited entry.
4. Some of these auto-cited entries have the 'dataonly' option set on them so that BibLaTeX will only use them for data and will not output them to the bibliography:
   - Clones of entries mentioned as a 'related' entry of a cited entry.

[7] A rather tricky unbounded loop but with a guaranteed eventual stable exit state.

[8] This is a complex, arbitrary multi-field Schwartzian Transform which has to deal with potentially different case and order settings for every field.

[9] This is done to implement the 'related' entry feature which does not currently have a BibLaTeX interface.

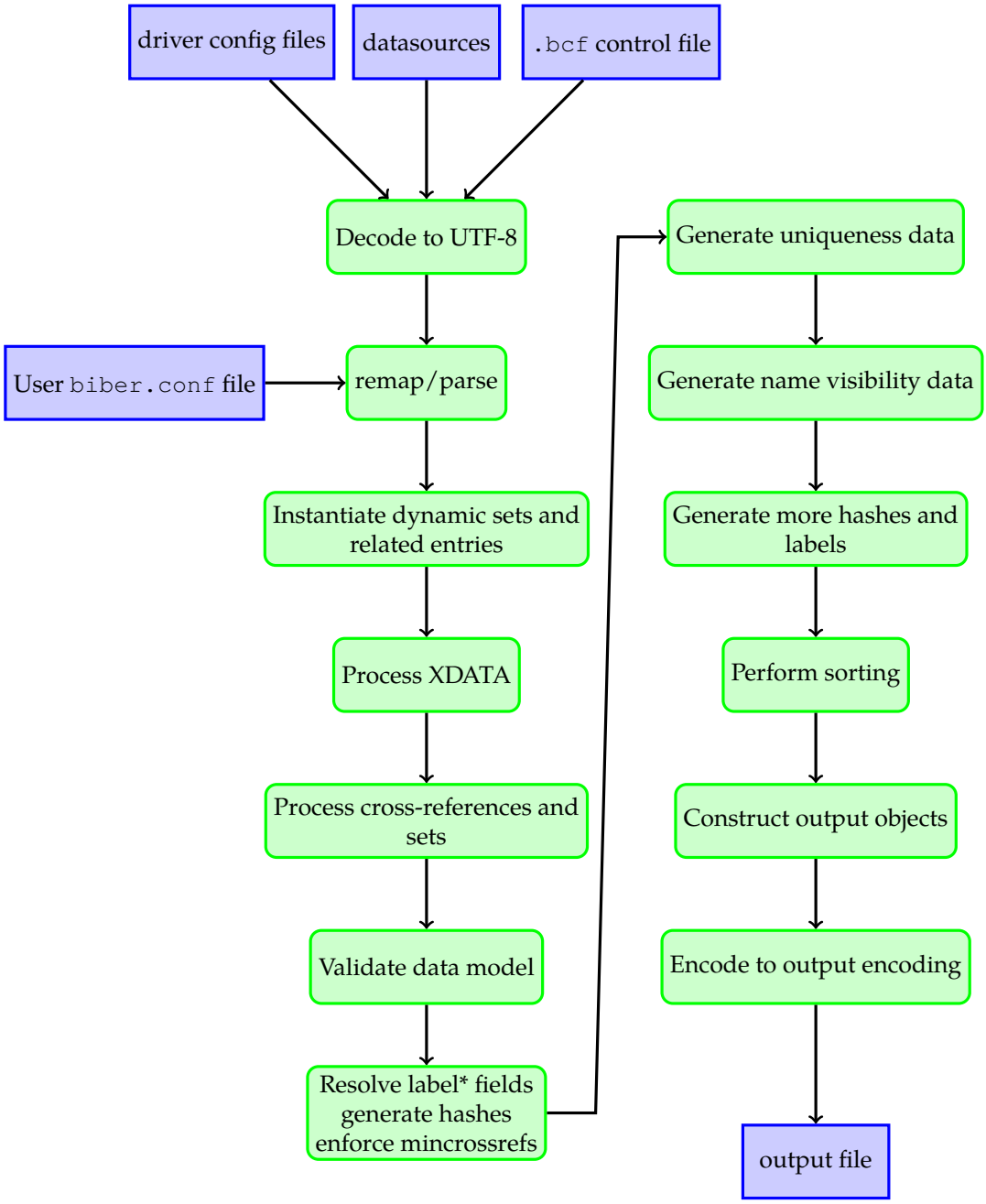


Figure 1: Overview of Biber's main functional units

### 2.6 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 under 3 minutes to process on on a reasonable computer. This is perfectly acceptable, especially for a batch program.

### 2.7 Acknowledgements

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

## 3 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.

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`[10]. 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

[10] BibLATEX Control File

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. BibLaTeX always writes the control file with a `.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
```

## 3.1 Options and config file

BibLaTeX options which Biber needs to know about are passed via the `.bcf` file. See Table 2 for the BibLaTeX options which Biber uses and also for the scopes which are supported for each option. Biber also has its own options which are set using the following resource chain, 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 via 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.

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 is XML. Here Below is an example config file which displays the Biber defaults:

| BibLaTeX option | Global | Per-type | Per-entry |
|---|---|---|---|
| alphaothers | ✓ | ✓ | |
| dataonly | | ✓ | ✓ |
| inheritance | ✓ | | |
| labelalpha | ✓ | ✓ | |
| labelalphatemplate | ✓ | ✓ | |
| labelnamespec | ✓ | ✓ | |
| labelnumber | ✓ | ✓ | |
| labelyear | ✓ | ✓ | |
| labelyearspec | ✓ | ✓ | |
| maxalphanames | ✓ | ✓ | ✓ |
| maxbibnames | ✓ | ✓ | ✓ |
| maxcitenames | ✓ | ✓ | ✓ |
| maxitems | ✓ | ✓ | ✓ |
| minalphanames | ✓ | ✓ | ✓ |
| minbibnames | ✓ | ✓ | ✓ |
| mincitenames | ✓ | ✓ | ✓ |
| minitems | ✓ | ✓ | ✓ |
| presort | ✓ | ✓ | ✓ |
| singletitle | ✓ | ✓ | |
| skipbib | | ✓ | ✓ |
| skiplab | | ✓ | ✓ |
| skiplos | | ✓ | ✓ |
| sortalphaothers | ✓ | ✓ | |
| sortexclusion | | ✓ | |
| sorting | ✓ | | |
| sortlos | ✓ | | |
| structure | ✓ | | |
| uniquelist | ✓ | ✓ | ✓ |
| uniquename | ✓ | ✓ | ✓ |
| useauthor | ✓ | ✓ | ✓ |
| useeditor | ✓ | ✓ | ✓ |
| useprefix | ✓ | ✓ | ✓ |
| usetranslator | ✓ | ✓ | ✓ |

Table 2: BibLaTeX options which Biber uses

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <bblencoding>UTF-8</bblencoding>
  <bibencoding>UTF-8</bibencoding>
  <bblsafechars>0</bblsafechars>
  <bblsafecharsset>base</bblsafecharsset>
  <bltxml>0</bltxml>
  <collate>1</collate>
  <collate_options>
    <option name="level" value="4"/>
  </collate_options>
  <debug>0</debug>
  <decodecharsset>base</decodecharsset>
  <fastsort>0</fastsort>
  <graph>0</graph>
  <mincrossrefs>0</mincrossrefs>
  <nodieonerror>0</nodieonerror>
  <nolog>0</nolog>
  <nostdmacros>0</nostdmacros>
  <nosort>
    <!-- strip prefices like 'al-' when sorting name fields -->
    <option name="type_names" value="\A\p{L}{2}\p{Pd}"/>
    <!-- strip diacritics when sorting name fields -->
    <option name="type_names" value="[\x{2bf}\x{2018}]"/>
  </nosort>
  <onlylog>0</onlylog>
  <quiet>0</quiet>
  <sortcase>true</sortcase>
  <sortlocale>en_US.utf8</sortlocale>
  <sortupper>true</sortupper>
  <trace>0</trace>
  <validate_config>0</validate_config>
  <validate_control>0</validate_control>
  <validate_structure>0</validate_structure>
  <wraplines>0</wraplines>
</config>
```

In practice, the most commonly used options will be set via BibLaTeX macros in your document and automatically passed to Biber via the `.bcf` file. Certain options apply only to Biber and can only be set in the config file, particularly the more complex options. Most options are simple tags. Exceptions are the `nosort` and `collate_options` options which are slightly more complex and can have sub-options as shown. A much more complex option is the `sourcemap` option which is not set by default and which is described in section 3.1.1.

### 3.1.1 The `sourcemap` option

The supplied data source drivers implement a default mapping from data source entrytypes and fields into the BibLaTeX data model[11]. If you want to override or augment the driver mappings you can use the `sourcemap` 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 other entrytypes/fields without modifying your data source. Essentially, this alters the source data stream which Biber uses to build the internal BibLaTeX data model. So, you are not constrained such that you must map into the BibLaTeX data model. Of course, if you don't, it's likely that your exotic new fields will be ignored by Biber because they are not valid in the BibLaTeX data model. In BibLaTeX 2.x, you will be able to re-define the internal BibLaTeX data model and pass this to Biber which will use it internally. Figure 2 is a graphical overview of the data flow for data model information. In BibLaTeX 2.x, the greyed static BibLaTeX data model will be replaced by a dynamic data model read from the `.bcf` control file. See Figure 1 for a more complete overview of Biber's processing steps.

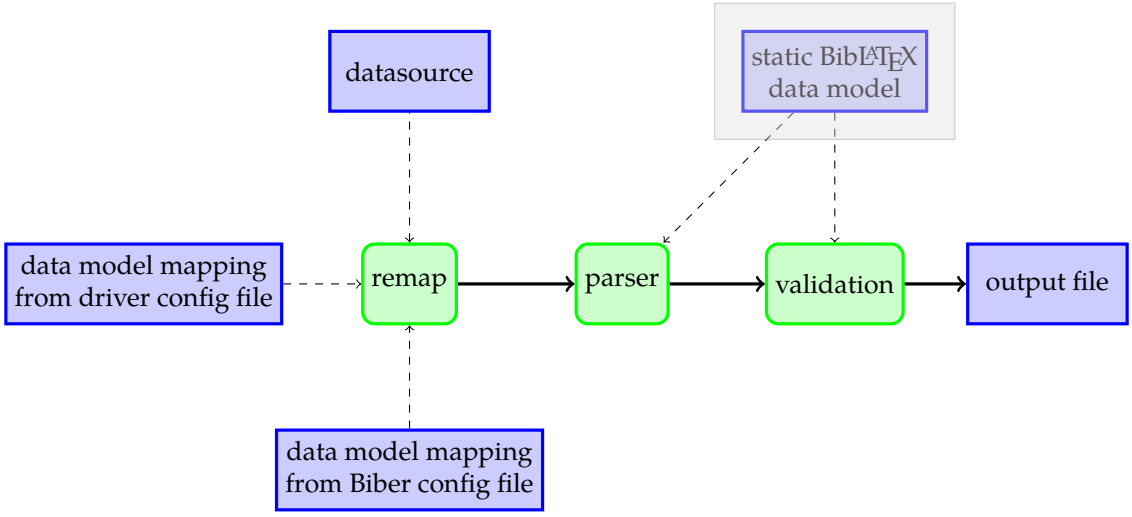


Figure 2: Model data flow in Biber

The `sourcemap` 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 tasks which can be useful for pre-processing data which you do not generate yourself:

[11] The BibLaTeX data model is currently static and defined in the BibLaTeX manual. In the future (BibLaTeX 2.x), the data model will be dynamically defined by the user.

- Map data source entrytypes to different entrytypes, optionally also adding new fields to the entry.
- Map data source fields to different 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.
- Modify the contents of a field using standard Perl regular expression match and replace.
- Restrict mappings to entries coming from particular data sources which you defined in `\addresource{}` macros.

The format of the `sourcemap` option section in the config file is described below, followed by examples which will make everything clear. 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 general structure is:

```
<sourcemap>
  <maps datatype="driver_1" map_overwrite="1|0">
    <map_1 maptype="entrytype|field" map_overwrite="1|0"> ... </map_1>
                ⋮
    <map_n maptype="entrytype|field" map_overwrite="1|0"> ... </map_n>
  </maps>
        ⋮
  <maps datatype="driver_n" map_overwrite="1|0">
    <map_1 maptype="entrytype|field" map_overwrite="1|0"> ... </map_1>
                ⋮
    <map_n maptype="entrytype|field" map_overwrite="1|0"> ... </map_n>
  </maps>
</sourcemap>
```

Here, $\texttt{driver}_1$...$\texttt{driver}_n$ are the names of valid Biber data source drivers (see section 3.8). One thing to note here is the `map_overwrite` attribute. This must be present and have a value of `0` or `1`. It determines whether, for this driver mapping section, whether you may overwrite existing fields when adding new fields or mapping them. This attribute can be overridden on a per-map basis, see below. A warning will be issued either way saying whether an existing field will or will not be overwritten. Only one mapping—the first mapping to match a given entry—will be used (since the mappings modify the entries).

There are two types of mappings you can apply to your datasources:

- Entrytype mapping
- Field mapping

An entrytype mapping allows you to do one or more of the following:

- Map an entrytype to another entrytype
- Add extra fields to all entries
- Add extra fields to entries of a given entrytype
- Restrict the changes to entries coming from particular datasources

A field mapping allows you to do one or more of the following:

- Map a field to another field
- Change the contents of a field
- Add extra fields to entries with particular fields
- Restrict the changes to entries coming from particular datasources
- Restrict the changes to entries of a particular entrytype

These facilities are explained in more detail below, with examples.

### Entrytype mapping

```
<map maptype="entrytype" map_overwrite="0|1">
  <per_datasource>datasource</per_datasource>
  <map_pair map_source="source-entrytype"
            map_target="target-entrytype"/>
  <also_set map_field="set-field"
            map_value="set-value"
            map_null="1"
            map_origentrytype="1"/>
</map>
```

An `entrytype` element performs a mapping of one or more mapping 'pairs' of entrytypes. Each pair consists of a `map_pair` element with a `map_source` attribute with value `source-entrytype` and an optional `map_target` attribute with value `target-entrytype`. The target is optional as you might want to simply modify entries of a certain entrytype without changing the entrytype. The `source-entrytype` is mandatory and can be the string '`*`' which means 'all entrytypes'. Mappings for specific entrytypes override the generic '`*`' type. The rules for an entrytype mapping are:

- Change the `source-entrytype` to `target-entrytype`, if defined.
- If there are any `datasource`s named in `per_datasource` elements, this mapping only applies to entries coming from the named `datasource`s. There can be multiple `per_datasource` elements each specifying one of the datasource names given in a BibLaTeX `\addbibresource` macro.
- There can be one or more `also_set` elements which each specify a behaviour for an entry field. The `map_field` attribute is mandatory and specified the field `set-field` in the entry to map. One and only one of these other attributes are then mandatory:

- map_value — The set-field is set to set-value
- map_null — The field is mapped to null, that is, it is deleted from the input stream completely. This is useful for ignoring field in datasources which are badly formatted or which you don't even want BibLaTeX to see at all.
- map_origentrytype — The set-field is set to the value of the source-entrytype name.

Here are some examples:

```
<map maptype="entrytype">
  <per_datasource>example1.bib</per_datasource>
  <per_datasource>example2.bib</per_datasource>
  <map_pair map_source="*"/>
  <also_set map_field="KEYWORDS" map_value="keyw1, keyw2"/>
</map>
```

This would add a KEYWORDS field with value 'keyw1, keyw2' to entries of any entrytype which are found in either the examples1.bib or examples2.bib files. This assumes that the BibLaTeX source contains \addresource{example1.bib} and \addresource{example2.bib}.

```
<map maptype="entrytype" map_overwrite="0">
  <map_pair map_source="CHAT" map_target="CUSTOMA"/>
  <also_set map_field="TYPE" map_origentrytype="1"/>
</map>
```

Any CHAT entrytypes would become CUSTOMA entrytypes and would automatically have a TYPE field set to 'chat' unless the TYPE field already existed in the entry.

```
<map maptype="entrytype">
  <map_pair map_source="ARTICLE"/>
  <map_pair map_source="BOOK"/>
  <also_set map_field="ABSTRACT" map_null="1"/>
  <also_set map_field="NOTE" map_value="Auto-created this field"/>
</map>
```

Any ARTICLE or BOOK entrytypes would have their ABSTRACT fields removed and a NOTE field added with value 'Auto-created this field'.

### Field mapping

```
<map maptype="field" map_overwrite="0|1">
  <per_datasource>datasource</per_datasource>
  <per_type>entrytype</per_type>
```

```
  <map_pair map_source="source-field"
            map_target="target-field"/>
            map_match="match-regexp"
            map_replace="replace-regexp"
            map_null="1"/>
  <also_set map_field="set-field"
            map_value="set-value"
            map_null="1"
            map_origfield="1"
            map_origfieldval="1"/>
</map>
```

Field mappings are a little different to entrytype mappings as it is more likely that several mappings will apply to the same entry. When processing an entry, Biber first filters the field mappings:

- If there are any `datasource`s named in `per_datasource` elements, this mapping only applies to entries coming from the named `datasource`s. There can be multiple `per_datasource` elements each specifying one of the datasource names given in a BibLaTeX `\addbibresource` macro.
- If there are any `entrytype`s named in `per_type` elements, this mapping only applies to entries of the named `entrytype`s.

The remaining mappings are then applied in the same order as they appear in the config file. Each mapping operates on the entry which results from a previous mapping. So, you should sequence the `map` elements in a processing chain. Within a `map` element, it is also possible to have multiple processing steps which are also processed in order but it is good practice to keep processing within a `map` element restricted to steps which are independent of each other and do the sequencing with `map` elements.

Within a `map` element, there are one or more of two types of processing element: mapping 'pairs'[12] and field creation 'also set' specifications.

Each mapping pair consists of a `map_pair` element with a `map_source` attribute with value `source-field` and an optional `map_target` attribute with value `target-field`. The target is optional as you might want to simply modify certain fields without mapping to other fields or simply establish that this mapping element applies to a certain field. The `source-field` is mandatory. Mapping pairs work in the following way:

- Change the `source-field` to `target-field`, if defined.

[12] Thusly called as their basic form is a mapping between a pair of fields, the 'source' and 'target' fields.

- Remove the source-field if map_null is set.
- If map_match is defined but map_replace is not, only apply the mapping if the source-field matches map_match.
- Perform a Perl regular expression match and replace on the value of source-field if map_match and map_replace are defined. You may use (and almost certainly will want to use) parentheses for back-references in map_replace. Do not quote the regular expressions in any way—it's not necessary.

Field creation is performed using an also_set element which specifies extra fields to add to an entry. The details of these elements and their semantics is given below:

- The map_field attribute is mandatory and specifies the field set-field to set. One and only one of these other attributes are then mandatory:
  - map_value — The set-field is set to set-value
  - map_null — The field is mapped to null, that is, it is deleted from the input stream completely. This is useful for ignoring field in datasources which are badly formatted or which you don't even want BibLaTeX to see at all.
  - map_origfield—The set-field is set to the source-field name of the most recent map_pair element.
  - map_origfieldval—The set-field is set to the source-field value of the most recent map_pair element.

Here are some examples:

```
<map maptype="field">
  <map_pair map_source="ABSTRACT" map_null="1"/>
  <map_pair map_source="CONDUCTOR" map_target="NAMEA"/>
  <map_pair map_source="GPS" map_target="USERA"/>
</map>
```

This removes any ABSTRACT fields from any entry, changes CONDUCTOR fields to NAMEA fields and changed GPS fields to USERA fields

```
<map maptype="field">
  <per_datasource>examples.bib</per_datasource>
  <per_type>MISC</per_type>
  <per_type>UNPUBLISHED</per_type>
  <map_pair map_source="ADDENDUM" map_null="1"/>
  <map_pair map_source="NOTE" map_null="1"/>
</map>
```

Here we remove any ADDENDUM or NOTE fields in MISC or UNPUBLISHED entries coming from the examples.bib datasource.

```
<map maptype="field">
  <map_pair map_source="PUBMEDID" map_target="EPRINT"/>
  <also_set map_field="EPRINTTYPE" map_origfield="1"/>
  <also_set map_field="USERD" map_value="Some string of things"/>
</map>
```

This maps `PUBMEDID` fields to `EPRINT` fields, sets the `EPRINTTYPE` field to 'pubmedid' and also sets the `USERD` field to the string 'Some string of things'.

```
<map maptype="field">
  <per_datasource>examples.bib</per_datasource>
  <map_pair map_source="SERIES"
          map_match="\A\d*(.+)"
          map_replace="\L$1"/>
</map>
```

Here, the contents of the `SERIES` field have leading numbers stripped and the remainder of the contents lowercased.

```
<map maptype="field">
  <map_pair map_source="TITLE"
          map_match="Collected\s+Works.+Freud"/>
  <also_set map_field="KEYWORDS" map_value="freud"/>
</map>
```

Here, if the `TITLE` field matches a particular regexp, we set a special keyword so we can, for example, make a references section just for certain items.

```
<map maptype="field">
  <map_pair map_source="AUTHOR"
          map_match="Smith, Bill" map_replace="Smith, William"/>
  <map_pair map_source="AUTHOR"
          map_match="Jones, Baz" map_replace="Jones, Barry"/>
</map>
```

Here, we use multiple match/replace for the same field to regularise some inconstant name variants. Bear in mind that match/replace processing within a `map` element is sequential and the changes from a previous match/replace are already committed. As mentioned above, it's often clearer to sequence using `map` elements to avoid confusion. This example is fine as the matches are exlusive and therefore mutually idempotent.

```
<maps datatype="bibtex" bmap_overwrite="1">
  <map maptype="field">
    <map_pair map_source="AUTHOR" map_match="Doe,"/>
    <also_set map_field="SHORTAUTHOR" map_origfieldval="1"/>
```

```
    <also_set map_field="SORTNAME" map_origfieldval="1"/>
  </map>
  <map maptype="field">
    <map_pair map_source="SHORTAUTHOR" map_match="Doe,\s*J(?:\.|ohn)(?:[
-]*)(?:P\.|Paul)*" map_replace="Doe, John Paul"/>
    <map_pair map_source="SORTNAME" map_match="Doe,\s*J(?:\.|ohn)(?:[
-]*)(?:P\.|Paul)*" map_replace="Doe, John Paul"/>
  </map>
</maps>
```

An example of sequential processing. First the `AUTHOR` field is copied to both the `SHORTAUTHOR` and `SORTNAME` fields, overwriting them if they already exist. Then, these two new fields are modified to canonicalise a particular name, which presumably has some variants in the datasource.

#### Other datasource types

For data sources other than BibTEX, (e.g. `ris`, `endnotexml` and `zoterordfxml`), the source entrytypes and fields are usually very differently modelled and named. For example, here is how to drop `subject` fields from various entrytypes in Zotero XML RDF format data sources:

```
<maps datatype="zoterordfxml" map_overwrite="1">
  <map maptype="field">
    <per_type>journalArticle</per_type>
    <per_type>book</per_type>
    <per_type>bookSection</per_type>
    <map_pair map_source="dc:subject" map_null="1"/>
  </map>
</maps>
```

Or here, mapping journal articles into `REPORT` entries for Endnote XML format data sources within a particular data source.

```
<maps datatype="endnotexml" map_overwrite="1">
  <map>
    <per_datasource>endnote.xml</per_datasource>
    <map_pair map_source="journal Article" map_target="REPORT"/>
  </map>
</maps>
```

Or here, dropping the `N2` field from `RIS` datasources, which are commonly used for abstracts:

```
<maps datatype="ris" map_overwrite="1">
     <map maptype="field">
```

```
      <map_pair map_source="N2" map_null="1"/>
    </map>
</maps>
```

### 3.1.2 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 3.

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>
  <!-- strip prefices like 'al-' when sorting names -->
  <option name="type_names" value="\A\p{L}{2}\p{Pd}"/>
  <!-- strip diacritics when sorting names -->
  <option name="type_names" value="[\x{2bf}\x{2018}]"/>
</nosort>
```

and the BibTEX 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[13].

You may specify any number of `option` elements. 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`:

```
<nosort>
  <option name="title" value="\AThe\s+"/>
</nosort>
```

[13] 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 3: `nosort` option field type aliases

If you wanted to do this for all title fields listed in Table 3, then you would do this:

```
<nosort>
  <option name="type_title" value="\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.

#### 3.1.3 The `collate_options` option

The `collate_options` option has format similar to `nosort`. See Section 3.4 for details about the option, here is an example of a config file setting:

```
<collate_options>
  <option name="level" value="3"/>
  <option name="table" value="/home/user/data/otherkeys.txt"/>
</collate_options>
```

### 3.2 Input/Output File Locations

#### 3.2.1 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

#### 3.2.2 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.

## 3.3 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.

## 3.4 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 Perl `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[14]. Options are 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.

### 3.4.1 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.

## 3.5 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.

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

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

### 3.5.1 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. For information on the `--bblsafechars` option, see section 3.5.2.

#### 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.

### 3.5.2 LaTeX macro encoding

The opposite of decoding; converting UTF-8 characters into LaTeX macros. You can force this with the `--bblsafechars` option which will do a generally good job of making your `.bbl` plain ASCII. It can be useful in certain edge cases where your bibliography introduces characters which can't be handled by your main document. See section 3.5.1 above for an example such case.

A common use case for LaTeX macro encoding is when the bibliography data source is not ASCII but the `.tex` file is and so this case is automated for you: if the BibLaTeX option '`texencoding`' (which corresponds to the Biber option '`--bblencoding|-E`') is set to an ASCII encoding ('`ascii`' or '`x-ascii`') and '`--bibencoding|-e`' is not ASCII, Biber will automatically set `--bblsafechars`.

See also the `biber --help` output for the `--bblsafecharsset` and `--decodecharsset` options which can customise the set of conversion rules to use. The characters and macros which Biber maps during encoding and decoding are documented[15].

### 3.5.3 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

[15] https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/0.9.8/documentation/utf8-macro-map.html

`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`

## 3.6 Editor Integration

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

### 3.6.1 Emacs

Emacs has the powerful AUCTeX mode for editing TeX and running compilations. Updated files for AUCTeX 11.86 are available here:

http://sourceforge.net/projects/biblatex-biber/files/auctex-biber.zip

Drop the `.el` files in the `.zip` file over the ones in your AUCTeX installation tree, delete the corresponding `.elc` files and run `M-0 M-x byte-recompile-directory` and give the path of your AUCTeX main install directory where the new files reside. Hopefully these modifications will make it into the official AUCTeX distributions soon.

The additions augment AUCTeX in the following ways:

- Adds font-lock support for most BibLaTeX macros
- Auto-detects whether you are using Biber with BibLaTeX
- Can detect whether dependencies like data sources have changed without them being open in Emacs
- Understands BibLaTeX and Biber messages so that AUCTeX will prompt you with the best default command to run next when using `C-cC-c`

### 3.6.2 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 3.

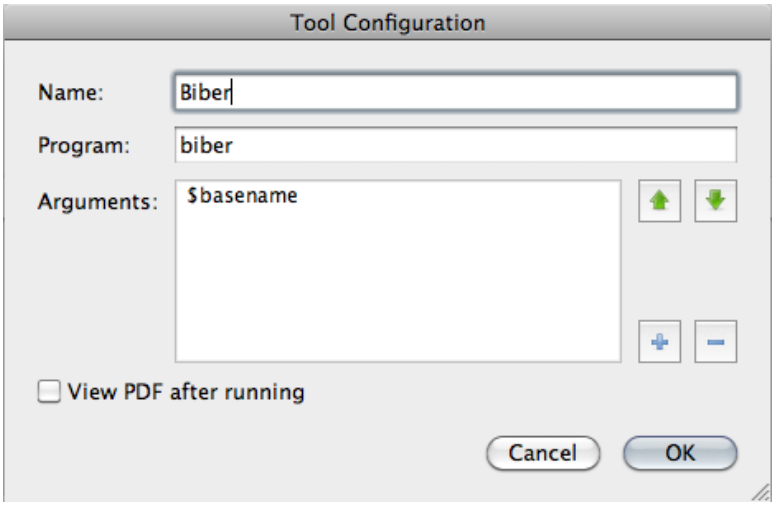


Figure 3: Screenshot of TEXworks processing tool setup for Biber

## 3.7 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 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.

## 3.8 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[16]. 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 meaningful information from a data source as possible. Currently supported data sources drivers are:

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

## 3.9 Visualising the Output

The option `--graph` will cause Biber to write a GraphViz[17] `.dot` file instead of a `.bbl`. This file graphs the bibliographic data as it exists after all processing. You can transform this file using the `dot` program from GraphViz to generate a high quality graphical representation of the data in a format of your choice. A good output format choice with `dot` is SVG[18] which can be viewed in any modern web browser. This format has the advantage of tooltips and Biber uses these to give you more information on connections between entries: hover the cursor on an arrow in the output and it will tell you what it means. To output in SVG, use this command after installing GraphViz:

```
dot -Tsvg <file>.dot -o <file>.svg
```

The `--graph` option takes a comma delimited string as argument. The elements of this string define the information to include in the `.dot` output graph. The valid sub-options are shown in Table 4. If the `--graph` option is given with no argument then the default is

```
--graph=crossref,section,xdata,xref
```

[16] https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/0.9.8/documentation/drivers
[17] http://www.graphviz.org
[18] Scalable Vector Graphics

| Sub-option | Description |
|---|---|
| crossref | Show crossreference relationships |
| field | Show fields within entries |
| related | Show related entries and clones |
| section | Show sections |
| xdata | Show XDATA relationships |
| xref | Show XREF relationships |

Table 4: Valid sub-options for the `graph` option

As with all options which take an optional value, be careful to terminate the options if you use `--graph` with the implied defaults, otherwise Biber will interpret the `.bcf` file name as the value of the option:

```
biber --graph -- file[.bcf]
```

Without the option terminating `--`, Biber would try to use the `.bcf` filename as the value of the `--graph` option. Figure 4 is a key to the elements included to the `.dot` output.

# 4 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.

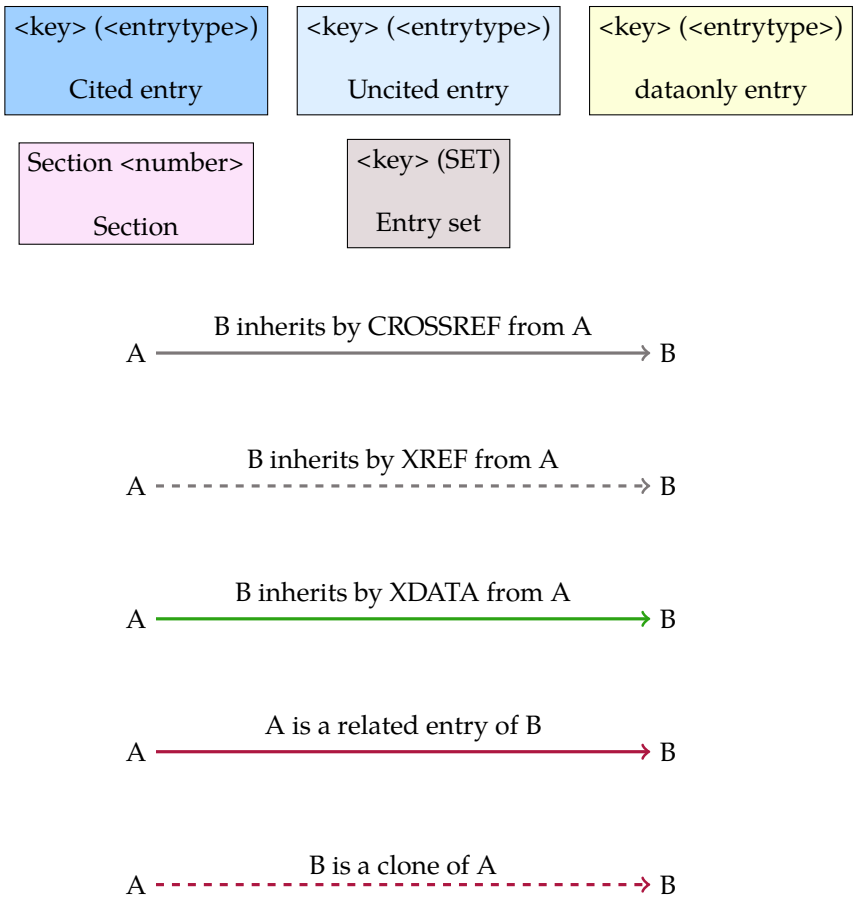


Figure 4: Key to `.dot` output format

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 4.1 below) and re-run the Biber executable again for the first time to allow it to unpack properly.

## 4.1 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/*/*/*/` (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.

## 4.2 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[19]
- freebsd_amd64[19]
- solaris_x86[19]

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.

## 4.3 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[20]. 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.

### 4.3.1 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.

[19] 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.

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

### 4.3.2 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 2.9\miktex\bin\
```

### 4.3.3 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/

## 4.4 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[21]:

- 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)

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

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.
- 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[22] for FAQs and help on building with PAR::Packer. Take special note of the FAQs on including libraries with the packed binary[23].

### 4.4.1 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[24], put them in a directory and run Biber in that directory like this:

```
biber --validate_control --convert_control test
```

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 0.9.8
INFO - Logfile is 'test.blg'
INFO - BibLaTeX control file 'test.bcf' validates
INFO - Converted BibLaTeX control file 'test.bcf' to 'test.bcf.html'
INFO - Reading 'test.bcf'
INFO - Found 1 citekeys in bib section 0
INFO - Processing bib section 0
INFO - Looking for BibTeX format file 'test.bib' for section 0
INFO - Found BibTeX data file 'test.bib'
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:

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

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

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

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