biber

A backend bibliography processor for biblatex

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

Version biber 2.11 (biblatex 3.11)
27th February 2018

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.

2.6

When outputting BibTeX data in tool mode (`--tool`), Biber now follows a full internal processing chain involving the data model. In previous versions, BibTeX output would just output the raw BibTeX input data, only allowing for some reformatting options and therefore no tool mode conversions from other formats into BibTeX format were possible. This change has some normalisation consequences:

- Dates are normalised into `DATE` fields. Legacy `YEAR` fields are never output in BibTeX format data output.
- Fields which are not defined in the data model described in the default `biber-tool.conf` are ignored and are neither read nor output. If custom fields are required, they should be defined in the data model by using a custom tool mode config file (see below). If you would like to have ignored fields reported on, use the `--validate-datamodel` option.

1.9

Biber no longer checks the environment for locales to use for sorting. This was always rather against the spirit of TeX since it means that the same document might look different when compiled by different people. However, Biblatex now passes Babel/Polyglossia language identifiers (or real locale identifiers if you prefer) in the `.bcf` and Biber can use these to set the sorting locale globally or on a per-sortscheme basis. This is better than using environment variables since Babel/Polyglossia are more LaTeX relevant language environments anyway.

1.8

Various option name changes. Old names are retained for backwards compatibility. See the output of the `--help` option.

1.0

The `--validate-structure` option is now called `--validate-datamodel`

0.9.9

The output format option `--graph` has been moved to a new option `--output-format`. The option `--graph` should now be specified as `--output-format=dot` and the `--dot-include` option should be used to specify the elements to include in the DOT output. For example:

```
biber --graph=section,field <file>
```

is now:

```
biber --output-format=dot --dot-include=section,field <file>
```

1.8

Several option names have changed. Several options have changed names to facilitate better semantic classification of options. The previous names are supported as legacy aliases. See the `--help` output of the Biber command.

0.9.8

The `sourcemap` option syntax has changed.The syntax was too confusing. It is now simplified and more powerful. It is uses a sequential processing model to apply mappings to an entry. See section 3.1.2.

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 3.1.2 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 to 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 primary role is to support Biblatex by performing the following tasks:

- Parsing data from datasources
- 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

Biber also has the ability to output different formats than `.bbl` and can, for example, output a new BibTeX file which contains only cited entries from the datasources (using the `--output-format=bibtex` option). There is also a '<tool' mode which operates on datasources instead of individual documents, allowing you to transform, convert, reformat and generally change the contents of a datasource (see 3.13).

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].

Biber's git repository and bug/feature tracker is on github[2]. Biber's documentation, binary downloads and supporting files are on SourceForge[3] Biber is included into TeXLive, the binaries coming from SourceForge.

2.3. Compatibility Matrix

Biber versions are closely 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 a Biblatex version which is incompatible. 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.

[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.

[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

Biber version	Biblatex version
2.11	3.11
2.10	3.10
2.9	3.9
2.8	3.8
2.7	3.7
2.6	3.5, 3.6
2.5	3.4
2.4	3.3
2.3	3.2
2.2	3.1
2.1	3.0
2.0	3.0
1.9	2.9
1.8	2.8x
1.7	2.7
1.6	2.6
1.5	2.5
1.4	2.4
1.3	2.3
1.2	2.1, 2.2
1.1	2.1
1.0	2.0
0.9.9	1.7x
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

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 3.12)
- 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 datasource architecture for ease of adding more datasource types
- Support for remote datasources
- User-definable mapping and suppression of fields and entrytypes in datasources. You can use this to, for example, ignore all `ABSTRACT` fields completely. See section 3.1.2
- Support for related entries, to enable generic treatment of things like 'translated as', 'reprinted as', 'reprint of' etc.

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

- Customisable labels
- Multiple bibliography lists in the same section with different sorting and filtering
- No more restriction to a static data model of specific fields and entrytypes
- Structural validation of the data against the data model with a customisable validation model
- Tool mode for operations on datasources directly

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.
   - 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 multi-field ST sort with an embedded cache for performance.

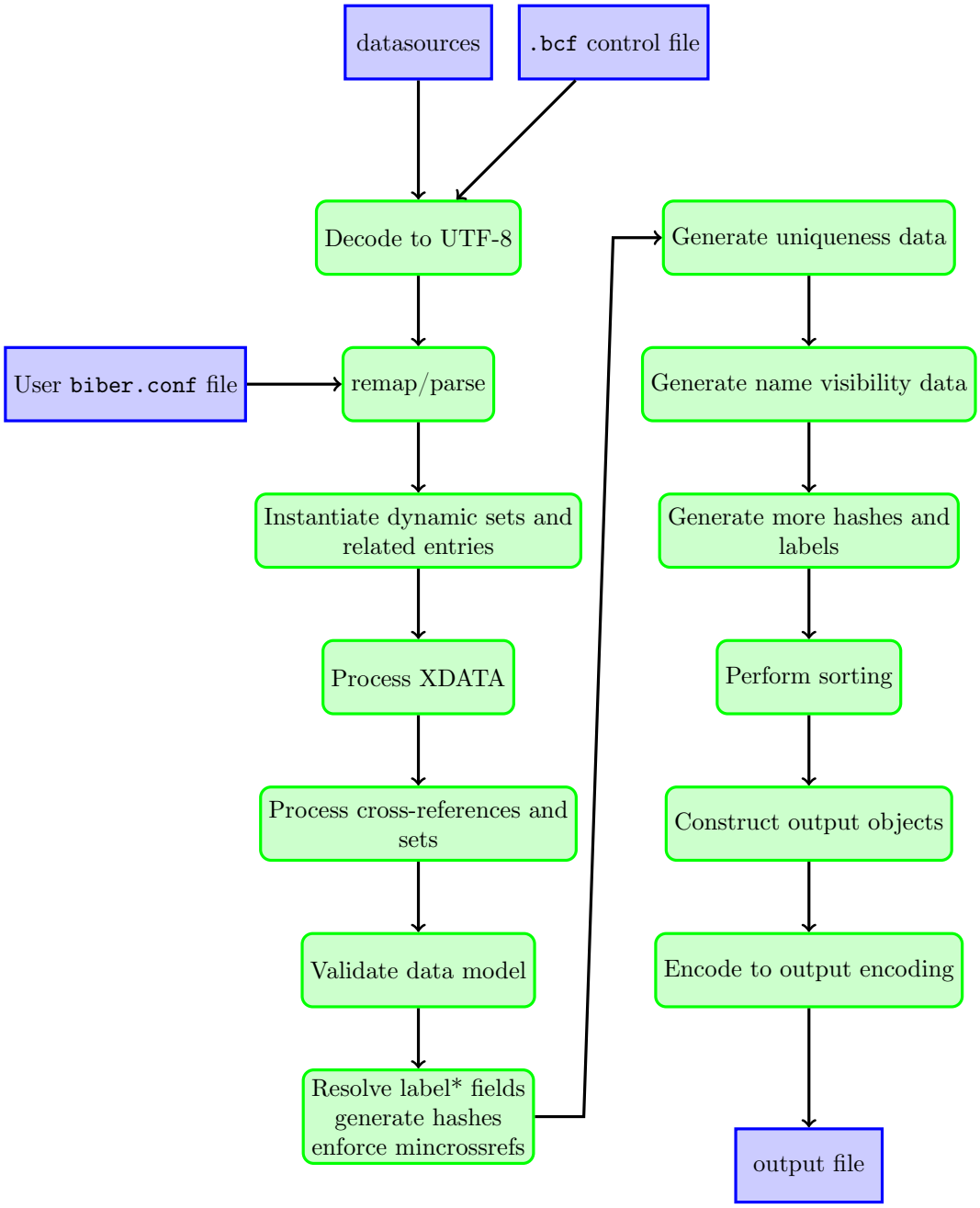

Figure 1: Overview of Biber's main functional units

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 just under 30 seconds to process on a reasonable computer. This is perfectly acceptable, especially for a batch program.

2.7. Acknowledgements

François Charette originally wrote a first modest version of Biber. Philip Kime joined in the development in 2009 and is largely responsible for making it what it is today.

3. Use

Firstly, please note that Biber will *not* attempt to sanitise the content of BibTeX datasources. 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 datasource types. In your BibTeX data sources, please make sure your entries are legal TeX code.

Running `biber --help` will display all options and description of each and is the primary 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`[9]. 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

[9] Biblatex Control File

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. 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
```

Figure 2 is a graphical overview of the data flow for data model information. See Figure 1 for a more complete overview of Biber's processing steps.

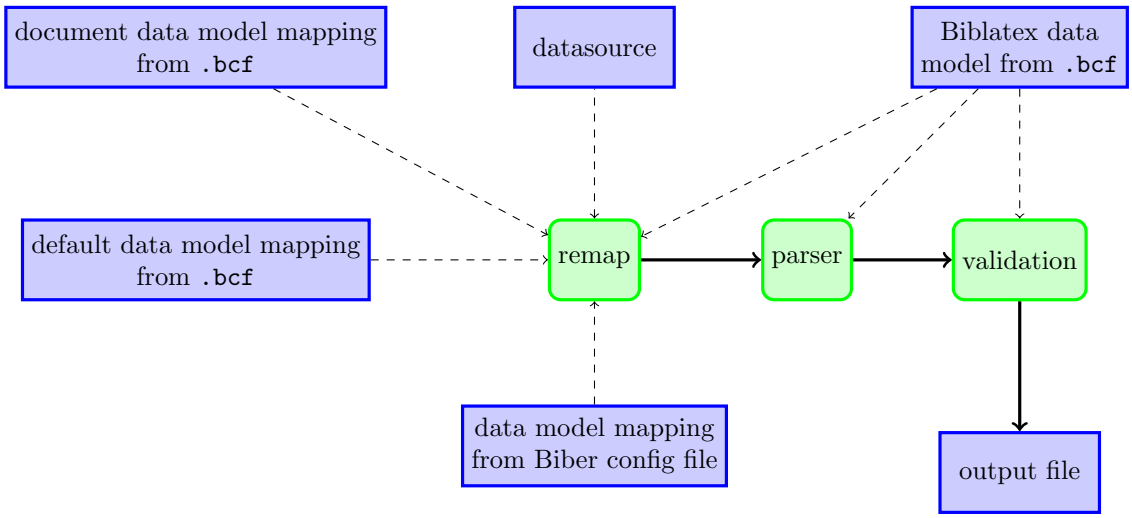

Figure 2: Model data flow in Biber

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→

Biblatex option	Global	Per-type	Per-entry
alphaothers	✓	✓	
dataonly		✓	✓
inheritance	✓		
labelalpha	✓	✓	
labelalphatemplate	✓	✓	
labeldate	✓	✓	
labeldatespec	✓	✓	
labelnamespec	✓	✓	
labelnumber	✓	✓	
labeltitle	✓	✓	
labeltitleyear	✓	✓	
maxalphanames	✓	✓	✓
maxbibnames	✓	✓	✓
maxcitenames	✓	✓	✓
maxitems	✓	✓	✓
minalphanames	✓	✓	✓
minbibnames	✓	✓	✓
mincitenames	✓	✓	✓
minitems	✓	✓	✓
presort	✓	✓	✓
singletitle	✓	✓	
skipbib		✓	✓
skiplab		✓	✓
skiplos		✓	✓
sortalphaothers	✓	✓	
sortexclusion		✓	
sortfirstinits	✓		
sorting	✓		
uniquelist	✓	✓	✓
uniquename	✓	✓	✓
useauthor	✓	✓	✓
useeditor	✓	✓	✓
useprefix	✓	✓	✓
usetranslator	✓	✓	✓

Table 2: Biblatex options which Biber uses

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:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <clrmacros>0</clrmacros>
  <collate_options>
    <option name="level" value="4"/>
    <option name="variable" value="non-ignorable"/>
    <option name="normalization" value="prenormalized"/>
  </collate_options>
  <debug>0</debug>
  <decodecharsset>base</decodecharsset>
  <dieondatamodel>0</dieondatamodel>
  <graph>0</graph>
  <input_encoding>UTF-8</input_encoding>
  <listsep>and</listsep>
  <mincrossrefs>0</mincrossrefs>
  <namesep>and</namesep>
  <nodieonerror>0</nodieonerror>
  <noinit>
    <!-- strip lowercase prefices like 'al-' when generating initials -->
    <option value="\b\p{Ll}{2}\p{Pd}"/>
    <!-- strip diacritics when generating initials -->
    <option value="[\x{2bf}\x{2018}]"/>
  </noinit>
```

```
  <nolabel>
    <!-- strip punctuation, symbols, separator and control characters -->
    <option value="[\p{P}\p{S}\p{C}]+"/>
  </nolabel>
  <nolog>0</nolog>
  <nostdmacros>0</nostdmacros>
  <nosort>
    <!-- strip prefices like 'El-' when sorting name fields -->
    <option name="setnames" value="\A\p{L}{2}\p{Pd}"/>
    <!-- strip some diacritics when sorting name fields -->
    <option name="setnames" value="[\x{2bf}\x{2018}]"/>
  </nosort>
  <onlylog>0</onlylog>
  <others_string>others</others_string>
  <ouput_align>0</output_align>
  <output_encoding>UTF-8</output_encoding>
  <output_fieldcase>upper</output_fieldcase>
  <output_format>bbl</output_format>
  <output_indent>2</output_indent>
  <output_resolve_xdata>0</output_resolve_xdata>
  <output_resolve_crossrefs>0</output_resolve_crossrefs>
  <output_resolve_sets>0</output_resolve_sets>
  <output_safechars>0</output_safechars>
  <output_safecharsset>base</output_safecharsset>
  <quiet>0</quiet>
  <sortcase>true</sortcase>
  <sortupper>true</sortupper>
  <tool>false</tool>
  <trace>false</trace>
  <validate_bltxml>0</validate_bltxml>
  <validate_config>0</validate_config>
  <validate_control>0</validate_control>
  <validate_datamodel>0</validate_datamodel>
  <wraplines>0</wraplines>
  <xsvsep>\s*,\s*</xsvsep>
</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`, `noinit` 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 3.1.2.

3.1.1. The `output-format` option

Biber is able to output formats other than `.bbl` files for Biblatex to consume. It is also able to output other formats such as `DOT` for visualisation of entry dependencies (see section 3.12), the experimental `biblatexml` XML format, BibTeX `.bib` files and an XML version of the `.bbl` format with extension `.bblxml`. `.bib` output is possible in tool mode, when you are converting an entire datasource file independently of any particular document (see section 3.13). It is also useful when you want, instead of a `.bbl`, a new `.bib` file containing only the cited entries from a document so that you can, for example, send a minimally complete package for typesetting to someone. To do this, you would, after the first LaTeX run, call Biber like this:

```
biber --output-format=bibtex test.bcf
```

This would result in a new `.bib` file called `test_biber.bib` containing all cited entries in `test.tex`, in citation order, formatted according to the various `ouput-*` options. You could of course also perform more processing like source mapping (see section 3.1.2), reencoding (see section 3.6) etc. using more command line options or a config file.

The `.bblxml` format for output is an XML version of the `.bbl`. It cannot be read by Biblatex but contains the same information as in the `.bbl` and may be useful if you want to transform a document bibliography into some other format since XML is a well-supported transformation format (using, for example, XSLT). By default, when choosing `.bblxml` output with the option `--output-format=bblxml`, a RelaxNG XML schema is also generated (unless the `--no-bblxml-schema` is used). This schema is derived from the active datamodel in the document (passed in the `.bcf` from Biblatex) and is placed in the same directory as the `.bblxml` output file. The extension of the schema is `.rng`. The option `--validate-bblxml` may be used to validate the `.bblxml` against the schema.

3.1.2. The `sourcemap` option

The datasource drivers implement a mapping from datasource entrytypes and fields into the Biblatex data model. 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 datasource with non-standard entrytypes or fields and to have these automatically mapped into other entrytypes/fields without modifying your datasource. Essentially, this alters the source data stream which Biber uses to build the internal Biblatex data model and is an automatic way of editing the datasource as it is read by Biber.

Source mappings can be defined at different 'levels' which are applied in a defined order. See the Biblatex manual regarding these macros:

user-level maps defined with \DeclareSourcemap→
  user-level maps defined in the Biber config file (described below)→
    style-level maps defined with \DeclareStyleSourcemap→
      driver-level maps defined with \DeclareDriverSourcemap

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:

- Map datasource entrytypes to different entrytypes.
- Map datasource fields to different fields.
- Add new fields to an entry
- Remove fields from an entry
- Modify the contents of a field using standard Perl regular expression match and replace.
- Restrict any of the above operations to entries coming from particular data-sources which you defined in \addresource{} macros.
- Restrict any of the above operations to entries only of a certain entrytype.

There is in fact, more flexibility than the above suggests, examples will show this below. The format of the sourcemap option section in the config file is described below, followed by examples which will make things clearer. 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 or element. The general structure is:

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

Here, driver$_1$...driver$_n$ are the names of valid Biber data source drivers (see section 3.11). One thing to note here is the map_overwrite attribute. This boolean

attribute determines whether, for this driver mapping section, 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. If omitted, it defaults to '0'.

The `map` elements are processed in sequence and contain a number of `map_steps` which are also processed in sequence. Each `map_step` allows you to do a particular thing or combination of things:

- Change the entrytype of an entry
- Change the name of a field
- Add extra fields the entry
- Change the contents of a field

These facilities are explained in more detail below, with examples. A `map` element looks like this:

```
<map map_overwrite="0|1" map_foreach="loopval">
  <per_datasource>datasource</per_datasource>
  <per_type>entrytype</per_type>
  <per_nottype>entrytype</per_nottype>
  <map_step  map_type_source="source-entrytype"
             map_field_source="source-field"
             map_notfield="source-field"
             map_type_target="target-entrytype"
             map_field_target="target-field"
             map_match="match-regexp"
             map_nomatch="match-regexp"
             map_replace="replace-regexp"
             map_field_set="set-field"
             map_field_value="set-value"
             map_entry_new="newentrykey"
             map_entry_newtype="newentrykeytype"
             map_entry_entrytarget="newentrykey"
             map_append="1"
             map_null="1"
             map_entry_null="1"
             map_entry_clone="clonekey"
             map_origfield="1"
             map_origfieldval="1"
             map_origentrytype="1"
             map_final="1"/>
</map>
```

- If there are any datasources named in per_datasource elements, this mapping only applies to entries coming from the named datasources. There can be multiple per_datasource elements each specifying one of the datasource names given in a Biblatex \addbibresource macro.
- If there are any entrytypess named in per_type elements, this mapping only applies to entries of the named entrytypess.
- If there are any entrytypess named in per_nottype elements, this mapping only applies to entries not of the named entrytypess.
- The map_overwrite attribute can be used to override the value for this attribute set on the parent maps element. If omitted, it defaults to the parent maps attribute value.
- The map_foreach attribute loops over all \steps in this \map, setting the special variable $MAPLOOP to each of the comma-separated values contained in loopval. loopval can either be the name of a datafield set defined with Biblatex's \DeclareDatafieldSet, a datasource field which contains a comma-separated values list or an explicit comma-separated values list itself (and loopval is determined in that order). This allows the user to repeat a group of map_steps for each value of loopval. The special variable $MAPUNIQ may also be used in the map_steps to generate a random unique string. This can be useful when creating keys for new entries. The special variable $MAPUNIQVAL may be used the map_steps to refer to the value of the last random unique string generated with $MAPUNIQ.

Each map_step is looked at in turn and compared with the datasource entry being processed. A map_step works like this:

- If map_entry_new is set, a new entry is created with the entry key newentrykey and the entry type newentrykeytype given in the option map_entry_newtype. This entry is only in-scope during the processing of the current entry and can be referenced by newentrykey given as the value to map_entrytarget. In newentrykey, you may use standard Perl regular expression backreferences to captures from a previous map_match step.
- When a map_field_set step has map_entrytarget set to the entrykey of an entry created by map_entry_new, the target for the field set will be the map_entrytarget entry rather than the entry being currently processed. This allows users to create new entries and set fields in them.
- If map_entry_null is set, processing of the map immediately terminates and the current entry is not created. It is as if it did not exist in the datasource. Obviously, you should select the entries which you want to apply this to using prior mapping steps.
- If map_entry_clone is set, a clone of the entry is created with an entry key clonekey. Obviously this may cause labelling problems in author/year styles

etc. and should be used with care. The cloned entry is in-scope during the processing of the current entry and can be modified by passing its key as the value to map_entrytarget. In clonekey, you may use standard Perl regular expression backreferences to captures from a previous map_match step.
- Change the source-entrytype to target-entrytype, if defined. If map_final is set then if the entrytype of the entry is not source-entrytype, processing of this map immediately terminates.
- Change the source-field to target-field, if defined. If map_final is set, then if there is no source-field field in the entry, processing of this map immediately terminates.
- If map_notfield is used then only apply the step if the source-field does not exist.
- If map_match is defined but map_replace is not, only apply the step if the source-field matches map_match. You can use parentheses as usual to capture parts of the match and can then use these later when setting a map_field_value.
- map_notmatch is the same as map_match but with the logic reversed.
- 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 special (i.e. non-Perly) way—it's not necessary.
- If map_field_set is defined, then its value is set-field which will be set to a value specified by further attributes. If map_overwrite is false for this step and the field to set already exists then the map step is ignored. If map_final is also set on this step, then processing of the parent map stops at this point. If map_append is set, then the value to set is appended to the current value of set-field. The value to set is specified by a mandatory one and only one of the following attributes:
  - map_field_value — The set-field is set to set-value
  - map_null — The field is ignored, as if it did not exist in the datasource
  - map_origentrytype — The set-field is set to the most recently mentioned source-entrytype name.
  - map_origfield — The set-field is set to the most recently mentioned source-field name
  - map_origfieldval — The set-field is set to the most recently mentioned source-field value

With BibTeX datasources, you can specify the pseudo-field 'entrykey' for source-field which is the citation key of the entry. Naturally, this 'field' cannot be changed (used as set-field, target-field or changed using map_replace).

Note that for XML datasources like BibLaTeXML, the names of fields and entrytypes are matched in a case sensitive manner. For all other datasource types, entrytype and field name matching is case insensitive.

Here are some examples:

```
<map>
  <per_datasource>example1.bib</per_datasource>
  <per_datasource>example2.bib</per_datasource>
  <map_step map_field_set="KEYWORDS" map_field_value="keyw1, keyw2"/>
  <map_step map_field_source="ENTRYKEY"/>
  <map_step map_field_set="NOTE" map_origfieldval="1"/>
</map>
```

This would add a KEYWORDS field with value 'keyw1, keyw2' and set the NOTE field to citation key for the entry to all entries 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 map_overwrite="0">
  <map_step map_field_source="TITLE"/>
  <map_step map_field_set="NOTE" map_origfieldval="1"/>
</map>
```

Copy the TITLE field to the NOTE field unless the NOTE field already exists.

```
<map map_overwrite="0">
  <map_step map_field_source="AUTHOR" />
  <map_step map_field_set="SORTNAME" map_origfieldval="1" map_final="1"/>
  <map_step map_field_source="SORTNAME" map_match="\A(.+?)\s+and.*" map_replace="$1"/>
</map>
```

For any entry with an AUTHOR field, try to set SORTNAME to the same as AUTHOR. If this fails because SORTNAME already exists, stop, otherwise truncate SORTNAME to just the first name in the name list.

```
<map map_overwrite="0">
  <map_step map_type_source="CHAT" map_type_target="CUSTOMA" map_final="1"/>
  <map_step map_field_set="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 exists in the entry (because map_overwrite is false). This mapping applies only to entries of type @CHAT since the first step has map_final set and so if the map_type_source does not match the entry, processing of this map immediately terminates.

```
<map>
  <per_datasource>examples.bib</per_datasource>
  <per_type>ARTICLE</per_type>
```

```
  <per_type>BOOK</per_type>
  <map_step map_field_set="ABSTRACT" map_null="1"/>
  <map_step map_field_set="NOTE" map_field_value="Auto-created this field"/>
</map>
```

Any entries of entrytype `ARTICLE` or `BOOK` from the 'examples.bib' datasource would have their `ABSTRACT` fields removed and a `NOTE` field added with value 'Auto-created this field'.

```
<map>
  <map_step map_field_set="ABSTRACT" map_null="1"/>
  <map_step map_field_source="CONDUCTOR" map_field_target="NAMEA"/>
  <map_step map_field_source="GPS" map_field_target="USERA"/>
</map>
```

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

```
<map>
  <map_step map_field_source="PUBMEDID"
             map_field_target="EPRINT"
             map_final="1"/>
  <map_step map_field_set="EPRINTTYPE" map_origfield="1"/>
  <map_step map_field_set="USERD"
             map_field_value="Some string of things"/>
</map>
```

Applies only to entries with `PUBMED` fields and 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>
  <map_step map_field_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>
  <map_step map_field_source="TITLE"
             map_match="Collected\s+Works.+Freud"
             map_final="1"/>
  <map_step map_field_set="KEYWORDS" map_field_value="freud"/>
</map>
```

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

```
<map>
  <map_step map_field_source="LISTA" map_match="regexp" map_final="1"/>
  <map_step map_field_set="LISTA" map_null="1"/>
</map>
```

If an entry has a LISTA field which matches regular expression 'regexp', then it is removed.

```
<map>
  <map_step map_field_source="AUTHOR"
            map_match="Smith, Bill" map_replace="Smith, William"/>
  <map_step map_field_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.

```
<map map_overwrite="1">
  <map_step map_field_source="AUTHOR" map_match="Doe," map_final="1"/>
  <map_step map_field_set="SHORTAUTHOR" map_origfieldval="1"/>
  <map_step map_field_set="SORTNAME" map_origfieldval="1"/>
  <map_step map_field_source="SHORTAUTHOR"
              map_match="Doe,\s*J(?:\.|ohn)(?:[-]*)(?:P\.|Paul)*"
              map_replace="Doe, John Paul"/>
  <map_step map_field_source="SORTNAME"
              map_match="Doe,\s*J(?:\.|ohn)(?:[-]*)(?:P\.|Paul)*"
              map_replace="Doe, John Paul"/>
</map>
```

Only applies to entries with an AUTHOR field matching 'Doe,'. 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.

```
<map>
  <map_step map_field_source="TITLE" map_match="A Title" map_final="1"/>
  <map_step map_entry_null="1"/>
</map>
```

Any entry with a TITLE field matching 'A Title' will be completely ignored.

Other datasource types

For datasources other than BibTeX, (e.g. `biblatexml`), the source entrytypes and fields are usually very differently modelled and named.
Here we use a loop to apply a regular expression replacement to several fields:

```
<maps datatype="bibtex" level="user">
  <map map_overwrite="1" map_foreach="author,editor,translator">
    <map_step map_field_source="$MAPLOOP" map_match="Smith" map_replace="Jones"/>
  </map>
</maps>
```

3.1.3. The `inheritance` option

The `inheritance` option defines the inheritance rules for data inheritance between entries using, for example, BibTeX's `CROSSREF` field. The default setup for this is defined by Biblatex and is passed in the `.bcf` file. Defining inheritance rules in the Biber configuration file is rarely something you would want to do with one notably exceptional case being when using Biber in tool mode where you might want to 'materialise' special inheritance rules (see section 3.13). Here we define the format of the config file inheritance section, should you need to understand or modify it. 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 or element.

```
<inheritance>
  <defaults inherit_all="true|false" override_target="true|false">
    <type_pair source="source" target="target"
               inherit_all="true|false"
               override_target="true|false"/>
                        ⋮
  </defaults>
  <inherit>
    <type_pair source="source" target="target"/>
                ⋮
    <field source="source"
           target="target"
           skip="true|false"
           override_target="true|false"/>
                        ⋮
  </inherit>
                        ⋮
</inheritance>
```

- The `defaults` section specifies the default inheritance rules which are not otherwise covered by a specific `inherit` rule. `inherit_all` specifies that by default a `target` inherits all fields from a `source`. `override_target` specifies that by default an existing `target` field will be overwritten by a `source` field it already exists in the `target`. A `type_pair` element specifies the defaults for a particular `source` and `target` entrytype combination. `source` or `target` can take the value '`*`' which is a wildcard representing all possible entrytypes.
- An `inherit` element specifies how one or more `source` fields are inherited by one more `source`/`target` pairs which are specified in one or more `type_pair` elements within the same `inherit` element. `override_target` can be specified on a per-field basis as can the `skip` attribute which indicates that a particular `field` is not to be inherited by the `target`.

Here is an example:

```
<inheritance>
  <defaults inherit_all="true" override_target="false">
  </defaults>
  <inherit>
    <type_pair source="mvbook" target="inbook"/>
    <type_pair source="mvbook" target="bookinbook"/>
    <type_pair source="mvbook" target="suppbook"/>
    <type_pair source="book" target="inbook"/>
    <type_pair source="book" target="bookinbook"/>
    <type_pair source="book" target="suppbook"/>
    <field source="author" target="author"/>
    <field source="author" target="bookauthor"/>
  </inherit>
  <inherit>
    <type_pair source="*" target="inbook"/>
    <type_pair source="*" target="incollection"/>
    <field source="*" skip="true"/>
  </inherit>
</inheritance>
```

Here we can see that the default is to inherit all fields from the `source` and not to override existing `target` fields if they already exist. Then we see that for some combinations of `source`s and `target`s, the `AUTHOR` field is inherited from the `source` and also the `AUTHOR` field in the `source` is inherited as the `BOOKAUTHOR` field in the `target`.

The second `inherit` element says that `INBOOK` and `INCOLLECTION` entries never inherit the `INTRODUCTION` field from any `source`.

In general, it is probably best to copy the default Biblatex inheritance rules and modify them to your needs. See section 3.13.

3.1.4. The `noinit` option

The value of the `noinit` 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. `noinit` allows you to ignore parts of a name when generating initials. This is done using Perl regular expressions which specify what to ignore. You can specific multiple regular expressions and they will be removed from the name before it is passed to the initials generating system.

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):

```
<noinit>
  <!-- strip lowercase prefices like 'al-' when generating initials -->
  <option value="\b\p{Ll}{2}\p{Pd}"/>
  <!-- strip diacritics when generating initials -->
  <option value="[\x{2bf}\x{2018}]"/>
</noinit>
```

and the BibTeX datasource entry:

```
AUTHOR = {{al-Hasan}, ʿAlī},
```

the initials for the last name will be 'H' and not 'a-H'. The initial for the first name will be 'A' as the diacritic is also ignored. This is tricky in general as you cannot often determine the difference between a name with a prefix and a hyphenated name with only, say, two chars in the first part such as 'Ho-Pun'. You can adjust this option for your individual cases. By default, only lowercased prefices are looked for so as to avoid breaking things like 'Ho-Pun' where you want the initials to be 'H.-P.', for example. See the Perl regular expression manual page for details of the regular expression syntax[10].

3.1.5. The `nolabel` option

The value of the `nolabel` 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. `nolabel` allows you to ignore elements of a field when generating labels. This is done using Perl regular expressions which specify what to ignore. You can specific multiple regular expressions and they will be removed from a field before it is passed to the label generating system.

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

For example, this option can be used to ignore control, punctuation, symbol and separator characters when generation labels. Given (the default):

```
<nolabel>
  <!-- strip punctuation, symbols, separator and control characters-->
  <option value="[\p{P}\p{S}\p{C}]+"/>
</nolabel>
```

and the BibTeX datasource entry with default label generation definition (see Biblatex documentation for `\DeclareLabelalphaTemplate`):

```
AUTHOR = {O'Toole, Alexander},
```

Then the label for the name will be «OTo07» as the apostrophe is ignored by the label generation routine. See the Perl regular expression manual page for details of the regular expression syntax[11].

3.1.6. The `nolabelwidthcount` option

The value of the `nolabelwidthcount` 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. `nolabelwidthcount` allows you to ignore elements of a field when generating fixed-width substrings of labels. This is done using Perl regular expressions which specify what to ignore. You can specific multiple regular expressions and they will be removed from a field before it is passed to the label generating system.

For example, this option can be used to ignore punctuation characters when generating substrings for labels. Note that in this example we reset `nolabel` because by default this removes punctuation characters. Given:

```
<nolabel>
  <option value=""/>
</nolabel>
<nolabelwidthcount>
  <option value="\p{P}+"/>
</nolabelwidthcount>
```

and the BibTeX datasource entry with default label generation definition (see Biblatex documentation for `\DeclareLabelalphaTemplate`):

```
AUTHOR = {O'Toole, Alexander},
```

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

Then the label for the name will be «O'To07» as the apostrophe is ignored by the substring generation routine. See the Perl regular expression manual page for details of the regular expression syntax[12].

3.1.7. The `sorting` option

The `sorting` option defines the sorting rules for the bibliography lists. Biblatex allows multiple sorting specifications referenced by name as it can print bibliography information as many times as the user wishes with different filtering and sorting. This is normally handled by macros in Biblatex which write the XML sorting specification(s) to the `.bcf` file for Biber to read but there may be occasions (usually when using Biber in 'tool' mode (see section3.13) when you need to specify the global sorting specification directly in a Biber config file. This section documents the XML format of the sorting specification. 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 or element. See also the `nosort` option in section 3.1.8.

```
<sorting scheme="schemename">
  <presort type=type>string</presort>
  <sortexclusion type=type>
      <exclusion>field</exclusion>
                ⋮
  </sortexclusion>
  <sort order="n"
        final=1
        sort_direction="ascending|descending"
        sort_case="1|0"
        sort_upper="1|0">
    <sortitem order="m"
              substring_side="left|right"
              substring_width="int"
              pad_side="left|right"
              pad_width="int"
              pad_char="string">field|literal|citeorder</sortitem>
              ⋮
  </sort>
     ⋮
</sorting>
```

Sorting in Biber is a sophisticated procedure of building up a sorting object for an entry based on a sorting scheme template, the general form of which is shown above. The sorting routine first traverses every entry in the bibliography list and

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

generates a sorting object based on the sorting scheme. When this is done, it sorts the entries according to the sorting objects it has generated for each entry.

A sorting specification must be named with the schemename attribute. In 'tool' mode, this must be set to `tool`. Otherwise, it is a name referenced by a Biblatex refcontext `sorting` option. A sorting specification is comprised of a number of `sort` elements. Sorting is essentially a process of comparing whatever information is in the nth `sort` element collected for every entry (otherwise known as 'multi-field' sorting). Within a `sort` element, there can be any number of `sortitem` elements which describe what information to look for in the entry in order to construct this part of the sorting object; either a field, a literal string or the special 'citeorder' pseudo-field.

When generating the sorting information for an entry, within each `sort` element, the first `sortitem` to return a non-empty value for the bibliography entry is used and the rest of the `sortitem`s in the `sort` are skipped. A `sortitem` essentially looks for a piece of information in the entry and adds this to the sorting object. If it is looking for a `field`, then the field must exist in the entry. If it does not, the `sortitem` is skipped. If the `field` does exist, it is added to the sorting object for the entry after being modified by the attributes of the `sortitem`.

Once a `sortitem` has returned the contents of a `field`, you can use the substring_side (default 'left' if any other substring attributes are set) and substring_width (default '4' if any other substring attributes are set) attributes to truncate the contents of the `field` by reducing it to a substring taken from the left or right side of the string and of a number of (UTF-8) characters of your choice. You can also pad the `field` with repeated arbitrary characters on either side using the pad_side (default 'left' if any other pad attributes are set), pad_width (default '4' if any other pad attributes are set) and pad_char (default '0'—the digit zero if any other pad attributes are set) attributes.

A `sortitem` which is neither a known bibliography sorting field nor the special 'citeorder' string is treated as a literal string to add to the sorting object. Naturally, such a `sortitem` always 'finds' something to add to the sorting object and so it should never have any other `sortitem`s after it within the `sort` section as they will never be considered. The 'citeorder' `sortitem` value has a special meaning. It requests a sort based on the lexical order of the actual citations. For entries cited in Biblatex within the same citation command like:

```
\cite{one,two}
```

there is a distinction between the lexical order and the semantic order. Here 'one' and 'two' have the same semantic order but a unique lexical order. The semantic order only matters if you specify further sorting to disambiguate entries with the same semantic order. For example, this is the definition of the Biblatex `none` sorting scheme:

```
<sorting>
  <presort>mm</presort>
  <sort order="1">
    <sortitem order="1">citeorder</sortitem>
  </sort>
</sorting>
```

This sorts the bibliography purely lexically by the order of the keys in the citation commands. In the example above, it sorts entry 'one' before 'two'. However, suppose that you consider 'one' and 'two' to have the same order (semantic order) since they are cited at the same time and want to further sort these by year. Suppose 'two' has an earlier `YEAR` than 'one':

```
<sorting>
  <presort>mm</presort>
  <sort order="1">
    <sortitem order="1">citeorder</sortitem>
  </sort>
  <sort order="2">
    <sortitem order="1">year</sortitem>
  </sort>
</sorting>
```

This sorts 'two' before 'one', even though lexically, 'one' would sort before 'two'. This is possible because the semantic order can be disambiguated by the further sorting on year. With the standard Biblatex `none` sorting scheme, the lexical order and semantic order are identical because there is nothing further to disambiguate them. This means that you can use 'citeorder' just like any other `sortitem` value, choosing how to further sort entries cited at the same time (in the same citation command).

Both `sort` and `sortitem` elements have a mandatory `order` attribute which should start at '1' and increase for each further element. Order numbers for `sortitem` elements within a `sort` element always begin with '1' and don't increase between `sort` elements.

Once a `sortitem` element has added something to the sorting object (or all `sortitem` elements within a `sort` have been processed, regardless of whether anything was added to the sort object for the entry), some attributes are applied to the information added and the next `sort` element is processed. These attributes on the `sort` element further determine how any sorting specification added by the `sortitem` elements will be used in the sorting.

If the `sort` element has the `final` attribute set to '1', then if any `sortitem` within the `sort` returned a non-empty string to add to the sorting object, the construction of the sorting object for the entry ceases at this point and no more `sort` elements are processed. This is used typically to make sure that master sorting keys such as

those specified with the SORTKEY field, if found, are the only thing used to construct the sorting object. The sort element may further specify that the information at order 'n' should be sorted in ascending order or descending order (default 'ascending'), whether case should be considered when sorting (default depends on the Biber sortcase option which defaults to true) and whether uppercase characters should be sorted before lower (default depends on the Biber 'sortupper' option which defaults to true).

Finally, there are two special sorting section elements to consider. The presort element is mandatory and specifies a literal string to add to the very beginning of all sorting objects for all entries. This is useful when combined with the fact that you may specify an optional type attribute which specifies a particular entry type for the presort string specified. Using this mechanism, you can sort, for example, all ARTICLE entries before all BOOK entries and then all other types of entry:

```
<sorting>
  <presort type="article">aa</presort>
  <presort type="book">bb</presort>
  <presort>mm</presort>
    ⋮
</sorting>
```

This makes it easy to divide a bibliography by type of entry.

The optional sortexclusion element allows you to exclude fields from consideration by sortitem on a per-type basis. For example, if you wanted to ignore the YEAR field of any REPORT entry types because they are not reliably populated with data representing a year, you could do:

```
<sorting>
    ⋮
  <sortexclusion type="report">year</sortexclusion>
    ⋮
</sorting>
```

It is much easier to see how intuitive this all is if you look at a standard sorting scheme definition. Below is the default Biblatex sorting scheme which appears in the .bcf when you run Biblatex with no sorting option. This is fully documented and described in the Biblatex manual along with the LaTeX macros which generate this XML in the .bcf:

```
<sorting>
  <presort>mm</presort>
  <sort order="1">
    <sortitem order="1">presort</sortitem>
  </sort>
  <sort order="2" final="1">
```

```
    <sortitem order="1">sortkey</sortitem>
  </sort>
  <sort order="3">
    <sortitem order="1">sortname</sortitem>
    <sortitem order="2">author</sortitem>
    <sortitem order="3">editor</sortitem>
    <sortitem order="4">translator</sortitem>
    <sortitem order="5">sorttitle</sortitem>
    <sortitem order="6">title</sortitem>
  </sort>
  <sort order="4">
    <sortitem order="1">sortyear</sortitem>
    <sortitem order="2">year</sortitem>
  </sort>
  <sort order="5">
    <sortitem order="1">sorttitle</sortitem>
    <sortitem order="2">title</sortitem>
  </sort>
  <sort order="6">
    <sortitem order="1" pad_side="left" pad_width="4" pad_char="0">volume</sortitem>
    <sortitem order="2">0000</sortitem>
  </sort>
</sorting>
```

3.1.8. 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. Datasource field sets defined using `\DeclareDatafieldSet` in Biblatex are also recognised as valid values and so it is possible to specify `nosort` regular expressions for arbitrary sets of fields. Biblatex defines as standard two sets as shown in Table 3.

For example, this option can be used to ignore some 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="setnames" value="\A\p{L}{2}\p{Pd}"/>
```

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

Table 3: Default Biblatex datafield sets

```
  <!-- strip diacritics when sorting names -->
  <option name="setnames" value="[\x{2bf}\x{2018}]"/>
</nosort>
```

and the BibTeX datasource entry:

```
AUTHOR = {{al-Hasan}, ʿAlī},
```

the prefix 'al-' and the diacritic 'ʿ' 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>
```

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

```
<nosort>
  <option name="settitles" 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.

3.1.9. The `collate-options` option

The `collate-options` option has format similar to `nosort`. See Section 3.5 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. Unicode

Biber uses NFD UTF-8 internally. All data is converted to NFD UTF-8 when read. If UTF-8 output is requested (to `.bbl` for example), the UTF-8 will always be NFC.

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

3.3. Input/Output File Locations

3.3.1. 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 `--input-directory`, if specified→
    In the `--output-directory`, if specified→
      Relative to current directory→
        Using `kpsewhich`, if available

3.3.2. Data sources

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

Absolute filename →
  In the `--input-directory`, if specified→
    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 datasources (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.

3.4. 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 `--output-file|-O` option to specify the location to write the `.bbl` to in such cases.

3.5. 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 uses 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.

The locale used for collating a particular field in the bibliography 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` →
    Biblatex per-sortset `locale` option →
      Biblatex per-sortscheme `locale` option→
        Biblatex global sortlocale 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 most standard collation cases are covered by the standard UCA and many locales neither have nor need any special collation tailoring.

Biblatex passes sortscheme-specific sorting locales and its global sorting locale in the `.bcf`. Biber uses these locales automatically to tailor sorting at various levels of granularity (see Biblatex docs for the `\DeclareSortingScheme` macro). Biblatex can be configured to automatically pass as locale the language setting from Babel or Polyglossia in which case Biber tries to match this to a sensible locale. See the Appendix, section A.1 for the mapping. If you want to sort using a specific locale not listed in A.1, specify this locale exactly in your LaTeX source as the Biblatex `sortlocale` option, as the optional argument to `\DeclareSortingScheme` macro or as an optional argument to the Biblatex `\sort` macro according to the desired granularity. For example, if you want to use traditional Spanish for sorting a reference list, you need to specify `es_ES_trad` directly instead of using the 'spanish'

string because the Babel/Polyglossia 'spanish' language identifier by default maps to the modern `es_ES` locale (which doesn't include sort tailoring for 'ch' in Spanish).

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.

By default, Biber 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.

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.

Note: Biber sets the Unicode collation option 'variable' to 'non-ignorable'. Effectively, this means that punctuation is not ignored when sorting. The default setting is to ignore such 'variable weight' elements. Sorting bibliographies is slightly more specialised than collating general text and punctuation often matters. In case you want the UCA default behaviour, see examples. Since Biber always normalises into NFD when reading data in, no normalisation is requested with Unicode collation ('normalization' option is set to 'prenormalized' by default) as this saves some time.

3.5.1. 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 `.bcf` if Biber's `sortlocale` option is not used to override the `.bcf`

`biber --sortlocale=de_DE_phonebook`

Override any locale setting in the `.bcf`

`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.

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

`biber --collate-options="variable => 'shifted'"`
: Use the UCA default setting for variable weight punctuation (which is to ignore it for sorting, effectively).

3.6. Encoding of files

Biber takes care of re-encoding the datasource data as necessary. In normal use, Biblatex passes its `bibencoding` option value to Biber via the `.bcf` file and this corresponds to the Biber `--input-encoding|e` option. Biblatex also passes the value of its `texencoding` option (which maps to Biber's `--output-encoding|-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 datasource into UTF-8 if it is not UTF-8 already
2. Decodes LaTeX character macros in the datasource into UTF-8
3. Encodes the output so that the `.bbl` is in the encoding that `--output-encoding|-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 `--output-encoding|-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:

`--input-encoding|-e` and `--output-encoding|-E` →
  Biber config file →
    `.bcf` control file

3.6.1. LaTeX macro decoding

As mentioned above, Biber always converts as much as possible, including LaTeX character macros, into UTF-8. This is important for two main reasons. Firstly, this allows you to have, for example:

```
@BOOK{key1,
    Author = {\"{O}leg Smith}
}
@BOOK{key2,
    Author = {Öleg Smith}
}
```

Here, because Biber decodes the macros into UTF-8, it knows that both books are by the same author because it's clear that the names are now the same. Secondly, this allows Biber to output normalised latex macros when a user selects `--output-encoding=ascii` etc. This means that the many Biblatex comparison macros used in styles can deal with comparisons of fields containing macros reliably. The macro to UTF-8 conversion uses the decoding set specified with the `--decodecharsset`, see below. To disable all macro to UTF-8 conversion, you can specify the virtual 'null' set as a value for `--decodecharsset` or `output-safecharsset`. This effectively turns off macro to UTF-8 decoding or encoding respectively.

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` datasource, 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 XƎTEX or LuaTEX as these don't use or need `inputenc`. However, you can also try the `--output-safechars` option which will try to convert any UTF-8 chars into LaTeX macros on output. For information on the `--output-safechars` option, see section 3.6.2.

3.6.2. LaTeX macro encoding

The opposite of decoding; converting UTF-8 characters into LaTeX macros. You can force this with the `--output-safechars` 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 3.6.1 above for an example such case.

A common use case for LaTeX macro encoding is when the bibliography datasource 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 '`--output-encoding|-E`') is set to an ASCII encoding ('`ascii`' or '`x-ascii`') and '`--input-encoding|-e`' is not ASCII, Biber will automatically set `--output-safechars`.

Since Biber always decodes into UTF-8 internally, if the `--output-encoding|-E` option is not set to UTF-8, Biber will automatically replace any characters which will not encode in the output encoding with equivalent TeX macros. You will also receive a warning about this.

See also the `biber --help` output for the `--output-safecharsset` and `--decodecharsset` options which can customise the set of conversion rules to use.

The builtin sets of characters and macros which Biber maps during encoding and decoding are documented[15].

It is possible to provide a customised encode/decode mapping file using the `--recodedata` option. It must adhere to the format of the default data file for reencoding which is `recode_data.xml` located in the same Perl install directory as Biber's `Recode.pm` module. Of course it is easier to find this in the Biber source tree. It is most likely that if you want to use a custom mapping file, you would copy the default file and edit it, removing some things and perhaps defining some custom recoding sets for use with `--output-safecharsset` and `--decodecharsset`.

Be careful to classify the entries using the correct 'type' attribute in the XML file as this determines how the macro is treated by the code that does the replacement. Just copy a similar type of macro from the default recoding data file if you are adding new entries, which is unlikely as the file is quite comprehensive. There is only one other thing to note. The 'preferred' attribute tells Biber to use a specific LaTeX macro when mapping from UTF-8, just in case there is more than one mapping from UTF-8 for a particular character. It's unlikely you will need to use this.

3.6.3. Examples

`biber`
: Set `input-encoding` and `output-encoding` from the config file or `.bcf`

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

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

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

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

`biber --decodecharsset=full`
: Set `input-encoding` and `output-encoding` 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` datasource which you want to sort correctly

`biber --recodedata=/tmp/recode.xml --decodecharsset=special`
: Specify a user-defined reencoding data file which defines a new reencoding set 'special' and use this set for decoding LaTeX macros to UTF-8.

[15] https://sourceforge.net/projects/biblatex-biber/files/biblatex-biber/2.11/documentation/utf8-macro-map.html

`biber -u`
: Shortcut alias for `biber --input-encoding=UTF-8`

`biber -U`
: Shortcut alias for `biber --output-encoding=UTF-8`

3.7. List and Name Separators

With traditional BibTeX, the name and list field separator 'and' is hard-coded. The `btparse` C library and therefore Biber allows the use of any fixed string, subject to the same rules as 'and' (not at the beginning or end of the name/list, whitespace must surround the string etc.). This is settable using the options `listsep` and `namesep`, both of which default to the usual 'and'. You can also change the default final name in a list which implies 'et al'. In BibTeX, this is by default the English 'others' which is the Biber default also. Don't try to put any whitespace in these strings, this is ignored by `btparse` anyway. Perhaps you prefer your `.bib` in more obvious German—set `--namesep=und` and `--others-string=andere` and then you can do:

```
@BOOK{key,
  AUTHOR = {Hans Harman und Barbara Blaupunkt und andere},
}
```

Bear in mind that these are global settings and apply to all entries in the BibTeX data format read by Biber. Also bear in mind that this is very unportable as all BibTeX input/output programs rely on the hard-coded 'and' and 'others'. Hopefully this will change as these two hard-coded English terms look really rather bad in the context of multilingual bibliographies.

3.8. Extended Name Format

The parsing rules for BibTeX names are rather archaic and not suited to many international name formats. Biber supports an extended name format which allows explicit specification of the parts of names. This allows the use of custom name parts apart from the four standard BibTeX parts. Extended name formats are supported in all name fields and can be used along with the usual BibTeX name format. Recognition of extended name format can be disabled with the Biber option `--noxname` in case you do not need the extended format and the auto-detection causes problems with normal name parsing. The separator `=` which comes between the namepart names and values is customisable with the Biber option `--xnamesep`. Here is an example:

```
AUTHOR = {Hans Harman and Simon de Beumont}
AUTHOR = {given=Hans, family=Harman and given=Simon, prefix=de, family=Beumont}
```

These two name specifications are equivalent but the extended format explicitly names the parts. The supported parts are those specified by the Biblatex data mode constant `nameparts`, the default value of which is:

```
\DeclareDatamodelConstant[type=list]{nameparts}{prefix,family,suffix,given}
```

As with traditional BibTeX name parsing, initials are automatically generated but it is also possible to specify these explicitly:

```
AUTHOR = {given=Jean, prefix=de la, prefix-i=d, family=Rousse}
AUTHOR = {given={Jean Pierre Simon}, given-i=JPS}
```

Initials are specified by adding the suffix `-i` to the namepart name. Compound parts may be protected with braces:

```
AUTHOR = {given={Jean Pierre}}
```

If a namepart contains a comma, the whole namepart should be protected with quotes:

```
AUTHOR = {"family={Robert and Sons, Inc.}"}
```

Traditional BibTeX name formats and the extended form may be used together:

```
AUTHOR = {Hans Harman and given=Simon, prefix=de, family=Beumont}
```

Per-namelist and per-name options may be specified in the extended name format:

```
AUTHOR = {namelistopt=true and Hans Harman and
          given=Simon, family=Beumont, nameopt=true}
```

3.9. Editor Integration

Visit http://tex.stackexchange.com/questions/154751/ for a comprehensive overview on Biber integration in most editors.

3.10. 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 datasource files, you should probably make any `MONTH` fields be the numbers which Biblatex expects.

3.11. Biber datasource drivers

Biber uses a modular datasource driver model to provide access to supported datasources. The drivers are responsible for mapping driver entrytypes and fields to the Biblatex data model according to a data model specification in the Biblatex file `blx-dm.def`. The data model can be changed using Biblatex macros in case you would like to, for example, use your own entrytype or field names or perhaps have Biber do some validation on your datasources (see the Biblatex manual).

Data model mapping is an imprecise art and the drivers are the necessarily the most messy parts of Biber. Most datasource 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 datasource as possible. Currently supported datasources drivers are:

- BibTeX — BibTeX data files
- `biblatexml` — Experimental Biblatex XML format

3.12. Visualising the Output

The option `--output-format=dot` will cause Biber to write a GraphViz[16] `.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[17] 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 `--dot-include` 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 `--dot-include` option is not given then the default setting is implicitly used, which is:

```
--dot-include=crossref,section,xdata,xref
```

[16] http://www.graphviz.org
[17] 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 `dot-include` option

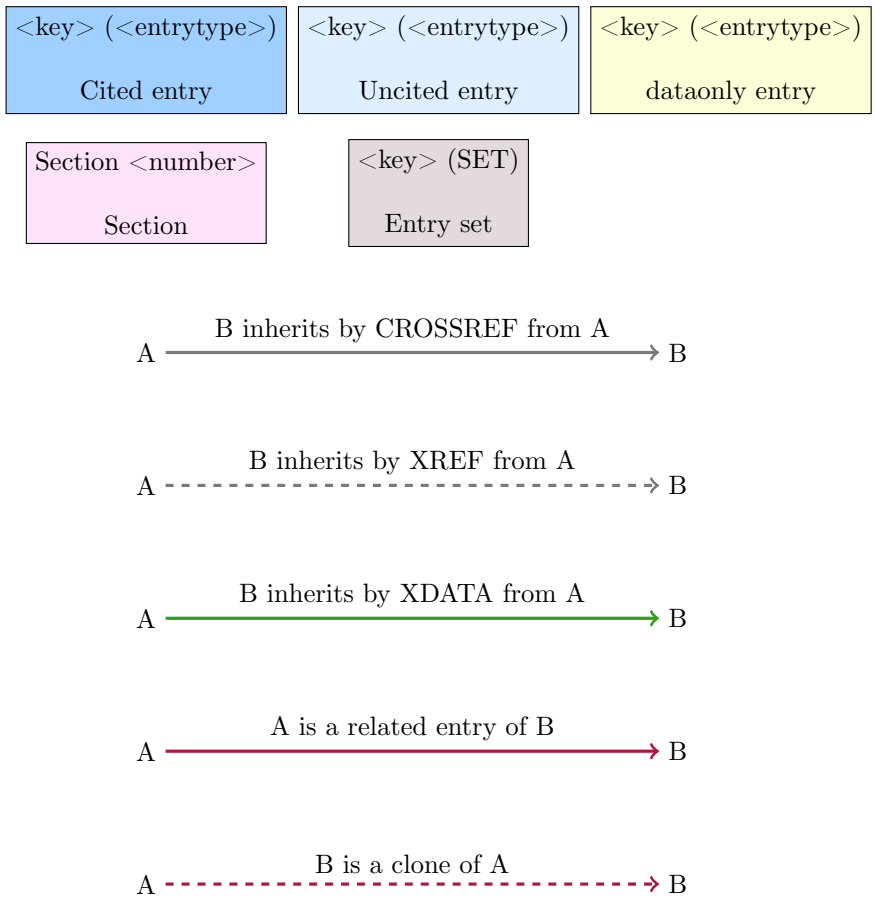

Figure 3: Key to `.dot` output format

3.13. Tool Mode

Biber can run in 'tool' mode which is enabled with the `--tool` command-line only option. In this mode, Biber is called: `biber --tool <datasource>`. Tool mode is a datasource rather than document oriented mode intended for transformations and modifications of datasources. It does not read a `.bcf` but instead, it reads all entries from the file 'datasource', applies any changes specified in the command-line options and Biber config file and writes the resulting datasource out to a new file, defaulting to a BibTeX file called '`<datasource>_bibertool.bib`' if the options `output-file` and `output-format` are not specified.

Tool mode is useful if you need to programatically change your datasource using the semantics provided by Biber or if you would like to convert your data to a different format. For example, you could choose to reencode your datasource by turning all UTF-8 characters into LaTeX macros:

```
biber --tool --output-encoding=ascii file.bib
```

This would output a copy of `file.bib` called `file_bibertool.bib` with all UTF-8 chars changed to LaTeX macros (because when the output is ASCII and the input encoding is not (it is by default UTF-8), then the `--output-safechars` option is automatically enabled). If you utilise the Biber config file, you can set up a complex set of mappings to transform your datasource however you wish in a semantic manner much more robust than just textual search/replace. You can also use the `--output-resolve` meta-option which will process any `XDATA` fields/entries, entry aliases and inheritance rules mentioned in the config file (see below).

Sometimes, you might wish to output fields which are BibTeX macros, that is, you might want this:

```
@Entrytype{key,
    Field = something,
}
```

instead of this:

```
@Entrytype{key,
    Field = {something},
}
```

That is, you might not want the output field in braces or quotes as this prevents BibTeX interpreting the field value as a macro. Use the `--output-macro-fields` option to specify a comma-separated list of fields whose values you wish to output without any BibTeX quoting. You can have spaces between the items in the field list but then you must enclose the whole option value in quotes. For example, these two will do the same thing:

```
biber --tool --output-macro_fields=month,publisher
biber --tool --output-macro_fields='month, publisher'
```

Tool mode also allows some reformatting of the `.bib` file. The option `--tool-fieldcase` can be used to force the entrytype and fieldnames to upper, lower or title case. The option `--tool-indent` can be used to customise the indentation of fields. The option `output-align` can be used to align all field values neatly. See the Biber `--help` output for documentation and defaults. For example, the command:

```
biber --tool --output-fieldcase=title --output-indent=4 \
      --output-align file.bib
```

results in `.bib` entries which look like this:

```
@Entrytype{key,
    Author    = {...},
    Title     = {...},
    Publisher = {...},
    Year      = {...},
}
```

another example:

```
biber --tool --output-fieldcase=upper --output-indent=2 file.bib
```

results in entries like this:

```
@ENTRYTYPE{key,
  AUTHOR = {...},
  TITLE = {...},
  PUBLISHER = {...},
  YEAR = {...},
}
```

Here is an example using the Biber config file to specify all options. This example uses tool mode to reformat the `.bib` and also to do some transformations using the source map functionality. Suppose `test.bib` contains the following:

```
@book{book1,
  author = {Doe,J.P.},
  title = {Ökologische Enterprises},
  year = {2013}
}
```

Further suppose that the `biber-tool.conf` contains the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <output_fieldcase>title</output_fieldcase>
  <output_encoding>ascii</output_encoding>
  <output_safechars>1</output_safechars>
  <sourcemap>
    <maps datatype="bibtex" map_overwrite="1">
      <map map_overwrite="1">
        <map_step map_field_source="AUTHOR" map_match="Doe," map_final="1"/>
        <map_step map_field_source="AUTHOR"
              map_match="Doe,\s*J(?:\.|ohn)(?:[-]*)(?:P\.|Paul)*"
              map_replace="Doe, John Paul"/>
      </map>
    </maps>
  </sourcemap>
</config>
```

Now you can run Biber like this:

```
biber --tool --configfile=biber-tool.conf test.bib
```

The result will be in `test_bibertool.bib` and will look like this:

```
@Book{book1,
  Author = {Doe, John Paul},
  Title  = {\"{O}kologische Enterprises},
  Year   = {2013},
}
```

Tool mode is a versatile way of performing many different operations on a `.bib` file. By using the config file and tool mode, we have:

- Consistently indented and aligned the entry, normalising fields and entrytype to title case
- Normalised the `AUTHOR` field name using regular expressions
- Converted UTF-8 characters to LaTeX macros, and made the output pure ASCII

If you do not specify any configuration file to use in tool mode, Biber will by default look for a config file in the usual way (see section 3.1) with the only difference that if no config file is found, it will use the default `biber-tool.conf` which is located in the Biber install tree in the same location as the `Config.pm` file. This default config file contains the default Biblatex source mappings for BibTeX datasources and also the default inheritance rules for `CROSSREF` processing. This means that when you use the `--output-resolve` meta-option, inheritance processing is performed on the entries and the results of this are 'materialised' in the output. For example, consider a `test.bib` file:

```
@BOOK{xd1,
  AUTHOR = {Edward Ellington},
  DATE   = {2007},
  XDATA  = {macmillanalias}
}

@XDATA{macmillan,
  IDS   = {macmillanalias},
  XDATA = {macmillan:pubALIAS, macmillan:loc}
}

@XDATA{macmillan:pub,
  IDS       = {macmillan:pubALIAS},
  PUBLISHER = {Macmillan}
}

@XDATA{macmillan:loc,
  LOCATION = {New York and London},
  NOTE     = {A Note}
}

@BOOK{b1,
  TITLE = {Booktitle},
  CROSSREF = {mvalias}
}

@MVBOOK{mv1,
  IDS = {mvalias},
  TITLE = {Maintitle},
  SUBTITLE = {Mainsubtitle},
  TITLEADDON = {Maintitleaddon}
}
```

Running Biber as:

```
biber --tool --output-resolve test.bib
```

The result of this would be a file `test_bibertool.bib` with contents:

```
@BOOK{xd1,
  AUTHOR    = {Edward Ellington},
  DATE      = {2007},
  LOCATION  = {New York and London},
  NOTE      = {A Note},
  PUBLISHER = {Macmillan},
}
```

```
@XDATA{macmillan,
  LOCATION  = {New York and London},
  NOTE      = {A Note},
  PUBLISHER = {Macmillan},
}

@XDATA{macmillan:pub,
  PUBLISHER = {Macmillan},
}

@XDATA{macmillan:loc,
  LOCATION = {New York and London},
  NOTE     = {A Note},
}

@BOOK{b1,
  MAINSUBTITLE   = {Mainsubtitle},
  MAINTITLE      = {Maintitle},
  MAINTITLEADDON = {Maintitleaddon},
  TITLE          = {Booktitle},
}

@MVBOOK{mv1,
  SUBTITLE   = {Mainsubtitle},
  TITLE      = {Maintitle},
  TITLEADDON = {Maintitleaddon},
}
```

Notice here that:

- `XDATA` references have been resolved completely for entry `xd1`
- `CROSSREF` inheritance has been resolved according to the default Biblatex inheritance rules for entry `b1`
- Entry key aliases have been resolved as required in order to perform these tasks

Tool mode can also be used to convert between datasource formats. For example, if you wish to covert a BibTeX format data file to the experimental `biblatexml` XML format, you can do:

```
biber --tool --output-format=biblatexml file.bib
```

This will output a file `file_bibertool.bltxml` by default. The applicability of the various output options depends on the output format as shown in table 5 where dash means that the options has no relevance for the output format.

Option	*Output format* bibtex	biblatexml	bbl	dot
output-align	✓	-	-	-
output-annotation-marker	✓	-	-	-
output-indent	✓	✓	-	-
output-field-order	✓	-	-	-
output-fieldcase	✓	-	-	-
output-listsep	✓	-	-	-
output-macro-fields	✓	-	-	-
output-namesep	✓	-	-	-
output-resolve-xdata	✓	✓	-	-
output-resolve-crossrefs	✓	✓	-	-
output-resolve-sets	✓	✓	-	-
output-xname	✓	-	-	-
output-xnamesep	✓	-	-	-

Table 5: Applicability of the output options

The order of the fields when writing BibTeX data is controlled by the `--output-field-order` option. This is a comma-separated list of fields or field classes and fields will be output to entries in the order specified. Any unspecified fields will be output in sorted order after the specified fields. The field classes are:

names All name fields
lists All non-name list fields
dates All date fields

For the default value, run Biber with the `--help` option and see the documentation for the option. `--output-listsep`, `output-namesep` and `output-xnamesep` can be used to customise separators on output and their default values are the same as their input option counterparts `--listsep`, `--namesep` and `--xnamesep`. The option `--output-xname` can be used to specify that the extended name format (see section **??**) is to be used to output names. `--output-annotation-marker` can be used to specify the annotation marker to write for annotated fields on output. See the Data Annotation feature documentation in the Biblatex manual.

3.13.1. Customising Tool Mode Inheritance Resolution

The default `biber-tool.conf` contains, as mentioned above, the default Biblatex `CROSSREF` inheritance setup and BibTeX source mappings so that tool mode resolution works as expected. Of course it is possible to customise these. In Biblatex, this

is accomplished by the `\DeclareDataInheritance` macros which write appropriate XML into the `.bcf` file. Since no `.bcf` file is used in tool mode, the desired configuration must be put into a Biber config file. The source mapping XML specification is given in section 3.1.2. The inheritance XML specification is given in section 3.1.3. It is recommended to copy the default `biber-tool.conf` file, modify this and then use it as your own `biber.conf` file or pass it explicitly using the `--configfile|-g` option. You can determine the location of the default tool mode config file by using the `--tool-config` option which will show you the location of the config file and exit.

3.13.2. Customising Tool Mode Sorting

A sorting scheme called 'tool' can be defined in the config file in order to sort the entries in tool mode output. See section 3.1.7 for the format of the config file sorting specification. By default, in tool mode the sorting scheme is the same as the Biblatex `none` scheme, that is, no sorting is performed. The sorting locale in tool mode defaults to '`en_US`' if you do not use Biber's `sortlocale` option.

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 5.24+ 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 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-<hex_encoded_username>/cache-<hash>` (Linux/Unix/OSX)
`par-<hex_encoded_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-<hex_encoded_username>` directory/folder and then run the current binary again. You can see the active cache by running biber with the `--cache` option which will print the current cache location and exit.

4.2. Binary Architectures

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

- `darwin_x86_64`
- `darwin_x86_i386`
- `linux_x86_32`
- `linux_x86_64`

- MSWin32
- MSWin64
- cygwin32[18]
- freebsd_x86[18]
- freebsd_amd64[18]
- solaris_i386[18]
- solaris_x86_64[18]

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[19]. 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.

[18] 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.

[19] 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 way 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 5.24+ with the following modules (best installed in this order):

- `Module::Build` and all dependencies
- All Biber pre-requisites
- `PAR::Packer` and all dependencies

Biber is very specific in some cases about module versions and sometimes depends on recent fixes. You can see if you have the Biber Perl dependencies by the usual `Module::Build` command:

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

run at the root of the Biber Perl distribution directory. Normally, the build procedure for the binaries is as follows[20]:

- 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)
- If you are missing dependencies, you will be informed and then you should run `Build installdeps` (may need to run this with sudo on Unix-like systems)

[20] On Unix-like systems, you may need to specify a full path to the scripts e.g. `./Build`

- Run the test suite with `Build test`
- Install with `Build install` (may need to run this with sudo on Unix-like 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 important 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 sub-folder in the `dist` directory named after the architecture you are building for.
- 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 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`[21] 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 missing library paths into `--link` options. 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

[21] `otool` on OSX and `depends.exe` on Windows

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 2.9
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 'nyt/global' keys
INFO - No sort tailoring available for locale 'en_GB.UTF-8'
INFO - Sorting list 'shorthands/global' 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
```

[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

There should now be these new files in the directory:

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

A. Appendix

A.1. Babel/Polyglossia language to Locale mapping

Language	Locale	Language	Locale	Language	Locale	Language	Locale
acadian	fr_CA	divehi	dv_MV	latin	la_Latn	sanskrit	sa_IN
american	en_US	dutch	nl_NL	latvian	lv_LV	scottish	gd_GB
australian	en_AU	english	en_US	lithuanian	lt_LT	serbian	sr_Latn
afrikaans	af_ZA	esperanto	eo_001	lowersorbian	dsb_DE	serbianc	sr_Cyrl
albanian	sq_AL	estonian	et_EE	lsorbian	dsb_DE	slovak	sk_SK
amharic	am_ET	ethiopia	am_ET	magyar	hu_HU	slovene	sl_SI
arabic	ar_001	farsi	fa_IR	malay	id_ID	slovenian	sl_SI
armenian	hy_AM	finnish	fi_FI	malayalam	ml_IN	spanish	es_ES
asturian	ast_ES	francais	fr_FR	marathi	mr_IN	swedish	sv_SE
austrian	de_AT	french	fr_FR	meyalu	id_ID	syriac	syc
bahasa	id_ID	frenchle	fr_FR	mongolian	mn_Cyrl	tamil	ta_IN
bahasai	id_ID	friulan	fur_IT	naustrian	de_AT	telugu	te_IN
bahasam	id_ID	galician	gl_ES	newzealand	en_US	thai	th_TH
basque	eu_ES	german	de_DE	ngerman	de_DE	thaicjk	th_TH
bengali	bn_BD	germanb	de_DE	nko	ha_NG	tibetan	bo_CN
bgreek	el_GR	greek	el_GR	norsk	nb_NO	turkish	tr_TR
brazil	pt_BR	hebrew	he_IL	nynorsk	nn_NO	turkmen	tk_TM
brazilian	pt_BR	hindi	hi_IN	occitan	oc_FR	ukrainian	uk_UA
breton	br_FR	ibygreek	el_CY	piedmontese	pms_IT	urdu	ur_IN
british	en_GB	icelandic	is_IS	pinyin	pny	UKenglish	en_GB
bulgarian	bg_BG	indon	id_ID	polish	pl_PL	uppersorbian	hsb_DE
canadian	en_US	indonesia	id_ID	polutonikogreek	el_GR	USenglish	en_US
canadien	fr_CA	interlingua	ia_FR	portuges	pt_PT	usorbian	hsb_DE
catalan	ca_ES	irish	ga_IE	portuguese	pt_PT	vietnamese	vi_VN
coptic	cop	italian	it_IT	romanian	ro_RO	welsh	cy_GB
croatian	hr_HR	japanese	ja_JP	romansh	rm_CH		
czech	cs_CZ	kannada	kn_IN	russian	ru_RU		
danish	da_DK	lao	lo_LA	samin	se_NO		