biber

A backend bibliography processor for biblatex

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

Version biber 0.6
4th November 2010

Contents

1 Introduction

1.1 About

`biber` is conceptually a `bibtex` replacement for `biblatex`. It is written in Perl with the aim of providing a customised and sophisticated data preparation backend for `biblatex`. Functionally, it 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.

1.2 Requirements

`biber` is distributed in two ways. There is a Perl source version which requires you to have a working Perl installation (preferably version 5.12 but no less than 5.10) and the ability to install the pre-requisite modules. Also provided are binaries for major OSes built with the Perl `PAR::Packer` module and utilities.

Currently there are binaries available for:

- OSX Intel 64-bit
- Windows
- Linux 32-bit

- Linux 64-bit

These should work on any fairly recent OS version. Both binaries and Perl source are available on SourceForge[1].

1.3 License

`biber` is released under the free software Artistic License 2.0[2]

1.4 History

`bibtex` has been the default (only …) integrated choice for bibliography processing in TeX for a long time. It has well known limitations which stem from its data format, data model and lack of Unicode support[3]. The `.bst` language for writing bibliography styles is painful to learn and use. It is not a general programming language and this makes it really very hard to do sophisticated automated processing of bibliographies.

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

`biber` was designed specifically for `biblatex` in order to provide a powerful backend engine which could deal with any required tasks to do with `.bbl` preparation. It can

- Deal with the full range of UTF-8
- Sort in a completely customisable manner, using when available, CLDR collation tailorings
- Automatically encode the `.bbl` into any supported encoding format[4]
- Process all bibliography sections in one pass of the tool
- Handle UTF-8 citekeys and filenames (given a suitable fully UTF-8 compliant TeX engine)
- Handle very complex auto-expansion and contraction of names and namelists.
- Lots of other things

[1] `http://sourceforge.net/projects/biblatex-biber/`
[2] `http://www.opensource.org/licenses/artistic-license-2.0.php`
[3] In fact, there is now a Unicode version
[4] 'Supported' here means encodings supported by the Perl `Encode` module

1.5 Acknowledgements

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

2 Use

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

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

```
biber --bibdata=somefile.bib
biber -d somefile.bib
```

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`[5]. This is conceptually equivalent to the `.aux` file which LaTeX uses to pass information to `bibtex`. The `.bcf` file is XML and contains many options and settings which configure how `biber` is to process the bibliography and generate the `.bbl` file.

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

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

2.1 Options and config file

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

[5] BibLaTeX Control File

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

Users do not need to care directly about the contents or format of the .bcf file as this is generated from the options which they specify for biblatex. To override the .bcf options or to provide option settings when no .bcf file is used (see for example the --allentries|-a and --bibdata|-d options), users may use either a configuration file or the command line to set options.

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

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

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

```
allentries          0
bibdata             undef
bblencoding         UTF-8
bibencoding         UTF-8
collate             1
<collate_options>
    level           3
</collate_options>
debug               0
fastsort            0
mincrossrefs        2
nolog               0
nosortdiacritics    [\x{2bf}\x{2018}]
nosortprefix        \p{L}{2}\p{Pd}
onlylog             0
```

```
quiet                0
sortcase             true
sortlocale           en_US.utf8
sortupper            true
trace                0
validate             0
wraplines            0
```

You can see here that options with multiple key/value pairs of their own like `--collate_options|-c` can be specified in Apache config format. Please see the documentation for the `Config::General` Perl module[6] if you really need details. In practise, if you use a config file at all for `biber`, it will contain very little as you will usually set all options by setting options in `biblatex` which will pass them to `biber` via the `.bcf` file.

The `--collate_options|-c` option takes a number of key/value pairs as value. See section 2.4 for details. The value of the `nosort*` options 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 anway. They specify stand-alone diacritic marks and name prefices to strip before sorting takes place and are designed to deal with cases like

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

where the prefix 'al-' and the diacritic 'ʿ' should not be considered when sorting.

2.2 Input/Output File Locations

2.2.1 Control file

The control file is normally passed as the only argument to biber (unless using `--allentries|-a` and `--bibdata|-d`). It is searched for using the following locations, in decreasing order of priority:

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

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

2.2.2 Database files

Bibliography database files are searched for using the same rule as for control files (see section 2.2.1 above). Unless using the `--bibdata|-d` option, users usually do not specify explicitly the bibliography database files; they are normally passed in the `.bcf` control file, taken from the `biblatex \bibliography{}` command arguments.

2.3 Logfile

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

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

then the logfile will be called '`test.blg`'. Like the `.bbl` output file, it will be created in the `--output-directory|-c`, if this option is defined. If there is no `.bcf` file on the command line (for example, when using the `--allentries|-a` and `--bibdata|-d` options), then the logfile name will default to '`biber.blg`'. You can override the logfile name by using the `--logfile` option:

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

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

2.4 Collation and Localisation

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

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

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

probably not a problem for you.

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

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

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

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

Collation is by default case sensitive. You can turn this off using the `biber` option `--sortcase=false` or from `biblatex` using its option `sortcase=false`.

`--collate|-C` by default collates uppercase before lower. You can reverse this using the `biber` option `--sortupper=false` or from `biblatex` by using its option `sortupper=false`.

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[7]. 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.

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

2.4.1 Examples

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

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

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

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

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

2.5 Encoding of files

`biber` takes care of reencoding the `.bib` data as necessary. In normal use, `biblatex` passes its `bibencoding` option value to `biber` via the `.bcf` file. It also passes an option `bblencoding` the value of which is derived from the `inputenc` package setting (if the user is using this package), otherwise 'utf8' (for XeTeX or LuaTeX) or 'ascii' (any other TeX engine).
`biber` performs the following tasks:

1. Decodes the `.bib` into UTF-8 if it is not UTF-8 already
2. Decodes LaTeX character macros into UTF-8
3. Encodes the output so that the `.bbl` is in the encoding that `bblencoding` specifies
4. Warns if it is asked to output to the `.bbl` any UTF-8 decoded LaTeX character macros which are not in the `bblencoding` encoding. Replaces with a diacritic-stripped substitute

As you can see from item 2 above, by default, `biber` converts LaTeX character macros into UTF-8 internally. This is very useful as it means that things are sorted correctly but has two potential (but rare) problems which you should be aware of:

- 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`, `biber` decodes this correctly to 'Đ' and this breaks `inputenc` because it doesn't understand that UTF-8 character. The solution here is to switch to a TeX engine with full UTF-8 support like XeTeX or LuaTeX as these don't use or need `inputenc`.

- If your `bblencoding` is not UTF-8, and you are using some UTF-8 equivalent LaTeX character macros in your `.bib`, then some `.bbl` fields (currently only `\sortinit{}`) might end up with invalid characters in them, according to the `.bbl` encoding. This is because some fields must be generated from the final sorting data which is only available after the LaTeX character macro decoding step.
  For example, suppose you were using PDFLaTeX with `\usepackage[latin1]{inputenc}` and the following `.bib entry`

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

  With normal LaTeX character macro decoding, the `{\v S}` is decoded into 'Š' and so with name-first sorting, `\sortinit{}` would be 'Š'. This is an invalid character in latin1 encoding and so the `.bbl` would be broken. In such cases when `\sortinit{}` is a char not valid in the `bblencoding`, `biber` strips off any diacritics which in this case results in 'S'. This is not ideal as this is not the initial character of the string used for sorting any more but it's a decent replacement in such cases. The solution is really to use UTF-8 `bblencoding` wherever possible. In extreme cases like the one above, this might also mean switching TeX engines to one that supports full UTF-8.

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 or config file. The resource chain for encoding settings is, in decreasing order of preference:

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

2.5.1 Examples

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

`biber --bblencoding=latin2`

Encode the `.bbl` as latin2, overriding the `.bcf`.

`biber -u`

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

`biber -U`

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

2.6 Limitations

Currently, users are restricted to the bibliography entry types hard-coded into `biblatex`. This is mitigated a little by the custom fields listed in section 2.2.4 of the `bibatex` manual but these are not portable or semantically obvious in their meaning. It is planned to have a customisable interface in `biblatex` which will allow users to define entry types an fields and have these passed through to `biber` which will validate the structure of the bibliography against these definitions. This would allow a fully customisable data model interface. Currently this is impossible due to a reliance on the `.bib` format which is quite restricted in scope and extensibility. It is likely that `biblatex` and `biber` will move to a modular data layer with an XML format as the default. `.bib` support will be maintained as a legacy format.

Currently it is not possible to automatically expand name lists to their minimally unique truncation which is required by some styles (APA for example). This is quite a hard problem, a solution to which is implemented in an experimental `biber` branch but which also needs `biblatex` support, envisaged for version 2.x. It requires an enhanced `.bbl` format, amongst other things.

2.7 Editor Integration

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

2.7.1 Emacs

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

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

and then `Ins` somewhere a new command that looks like Figure 1.
Alternatively, you can add it directly in lisp to your `.emacs` like this:

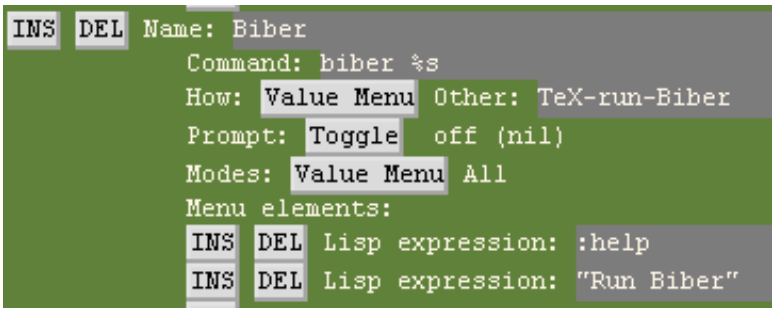

Figure 1: Screenshot of AUCTeX command setup for Biber

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

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

```
; Biber under AUCTeX
(defun TeX-run-Biber (name command file)
  "Create a process for NAME using COMMAND to format FILE with Biber."
 (let ((process (TeX-run-command name command file)))
    (setq TeX-sentinel-function 'TeX-Biber-sentinel)
    (if TeX-process-asynchronous
process
      (TeX-synchronous-sentinel name file process))))

(defun TeX-Biber-sentinel (process name)
  "Cleanup TeX output buffer after running Biber."
  (goto-char (point-max))
  (cond
   ;; Check whether Biber reports any warnings or errors.
   ((re-search-backward (concat
 "^(There \\(?:was\\|were\\) \\([0-9]+\\) "
 "\\(warnings?\\|error messages?\\))") nil t)
    ;; Tell the user their number so that she sees whether the
    ;; situation is getting better or worse.
    (message (concat "Biber finished with %s %s. "
     "Type `%s' to display output.")
     (match-string 1) (match-string 2)
```

```
    (substitute-command-keys
     "\\<TeX-mode-map>\\[TeX-recenter-output-buffer]")))
  (t
   (message (concat "Biber finished successfully. "
    "Run LaTeX again to get citations right."))))
 (setq TeX-command-next TeX-command-default))
```

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

2.7.2 TeXworks

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

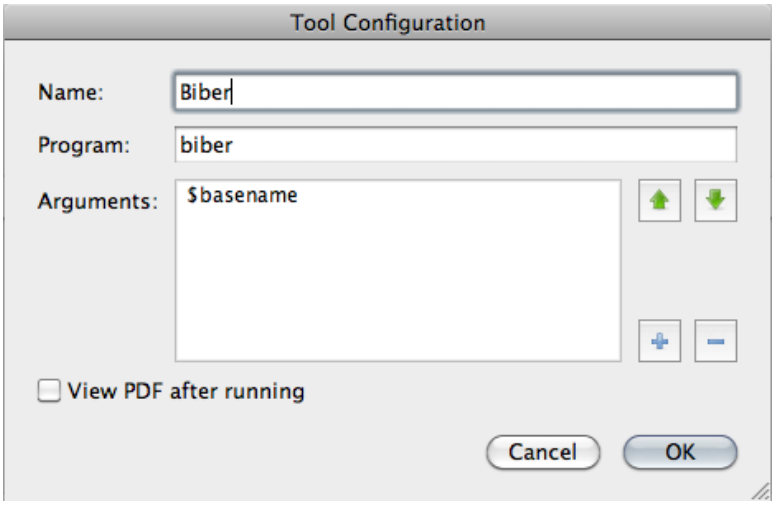

Figure 2: Screenshot of TeXworks processing tool setup for Biber

2.8 Cross-references

`biblatex` supports the `CROSSREF` field in bibliography databases and because `biber` performs the actual processing of such fields, it can do this in a customisable way. Currently, the customisable interface via user `biblatex` macros and the `.bcf` file is not yet implemented but the internal code in `biber` is ready for this. The purpose of this section is to describe the current hard-coded `biber` defaults. By default, the cross-reference target (child) will inherit all of the source (parent) fields if they are not already defined in the target. Table 1 shows the current custom

Source	Target	Source field	→	Target field
PROCEEDINGS	INPROCEEDINGS	TITLE	→	BOOKTITLE
		SUBTITLE	→	BOOKSUBTITLE
		TITLEADDON	→	BOOKTITLEADDON
COLLECTION	INCOLLECTION	TITLE	→	BOOKTITLE
		SUBTITLE	→	BOOKSUBTITLE
		TITLEADDON	→	BOOKTITLEADDON
BOOK	INBOOK	TITLE	→	BOOKTITLE
		SUBTITLE	→	BOOKSUBTITLE
		TITLEADDON	→	BOOKTITLEADDON
		AUTHOR	→	BOOKAUTHOR

Table 1: Custom cross-reference mappings

mappings which apply. These custom cross-reference mappings will overwrite the target fields, if they exist. In future, the fully customisable interface will allow:

- Choice of overwriting target fields with source fields
- Choice of whether or not to, by default, inherit all fields from the source
- Mapping of any source field to any target field
- Mapping of field(s) in any source to specific targets
- Mapping of field(s) in specific sources to any targets
- Many-to-one and one-to-many field mappings
- Skipping inheritance of specific source fields

The custom cross-reference mappings will make it possible to overcome the problems with the traditional `bibtex` handling of cross-references which are mentioned in section 2.4.1 of the `biblatex` manual. As mentioned above, only the hard-coded mappings in Table 1 are currently implemented.

3 Binaries

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 and so the binaries look larger than what actually runs in memory.
- On the first run of a new version (that is, with a specific checksum), 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.

3.1 Binary Architectures

Binaries are available for the following architectures:

- `linux_x86_32` — Linux x86 32-bit (built on Ubuntu 9.04)
- `linux_x86_64` — Linux x86 64-bit (built on Ubuntu 9.04)
- `MSWin` — Windows. Should work on 32 or 64 bit (built on XP)
- `darwin_x86_64` — OSX Intel 64-bit (built on OSX 10.6)

3.2 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[8]. Below I assume it's on your desktop.

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

3.2.1 OSX

If you are using the TexLive MacTeX distribution:

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

If you are using the macports TexLive distribution:

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

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

3.2.2 Windows

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

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

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

3.2.3 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/

3.3 Building

Instructions for those who want/need to build an executable from the Perl version. For this, you will need to have a recent Perl, preferably 5.12 at least with the following modules:

- `PAR`
- `PAR::Packer`
- All `biber` pre-requisites

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

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

invocation in the biber Perl distribution tree directory. Normally, the build procedure for the binaries is as follows[9]:

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

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

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

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

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

- Make a new subfolder in the `dist` directory named after the architecture you are building for. This name is arbitrary but should be fairly obvious like '`solaris-sparc-64`', for example.
- Copy the `biber.files` file from an existing build architecture into this directory.
- For all of the files with absolute pathnames in there (that is, ones we are not pulling from the `biber` tree itself), locate these files in your Perl installation tree and put the correct path in the file.
- Copy the build script from a vaguely similar architecture (i.e. Windows/non-Windows …) to your new architecture directory.
- Change the `--link` options to point to where the required libraries reside on your system.
- Change the `--output` option to name the resulting binary for your architecture.
- Run the build script

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